P. SIRVEN

LANGAGE MACHINE

POUR ZX 81



S. E. C. F.
ÉDITIONS RADIO

LANGAGE MACHINE
POUR ZX 81

P. SIRVEN

LANGAGE MACHINE

POUR ZX 81



9, RUE JACOB - 75006 PARIS
TEL. 329.63.70

Imprimé en France par Berger-Levrault, Nancy

Dépôt légal : juillet 1984
Éditeur n° 970 - Imprimeur : 778085
I.S.B.N. 2 7091 0946 8

PROLOGUE

Pour accéder au langage machine le micro-ordinateur ZX 81 dispose de trois instructions Basic : PEEK, POKE et USR, mais le manuel de programmation SINCLAIR qui est livré avec le micro-ordinateur ZX 81 ne traite que la programmation en Basic.

Dans l'annexe A de ce manuel on trouve cependant de nombreux éléments relatifs à la programmation en langage machine, comme la liste des instructions en code machine.

Le micro-ordinateur ZX 81 est équipé d'un microprocesseur Z 80 A ; c'est donc le langage machine de ce microprocesseur qui sera utilisé pour la programmation en langage machine du ZX 81, car chaque microprocesseur possède un langage machine différent.

Le langage machine d'un microprocesseur est introduit dans celui-ci lors de la fabrication du microprocesseur, contrairement aux langages évolués comme le Basic qui doivent faire partie de la mémoire centrale du micro-ordinateur.

La meilleure manière d'apprendre le langage machine est d'exécuter des programmes de difficultés croissantes dans ce langage, tout comme la meilleure manière d'apprendre le Basic consiste à exécuter des programmes Basic de plus en plus difficiles.

Il est possible de programmer un micro-ordinateur en Basic ou en tout autre langage évolué sans avoir aucune connaissance concernant la constitution interne de ce micro-ordinateur et même sans savoir quel type de microprocesseur équipe le micro-ordinateur.

Il n'en va plus de même lorsqu'on désire programmer ce micro-ordinateur en langage machine ; dans ce cas il faut posséder un minimum de connaissances concernant la mémoire centrale du micro-ordinateur et il faut aussi connaître le fonctionnement des registres et la liste des instructions en code machine du microprocesseur équipant le micro-ordinateur.

Les instructions constituant le langage machine d'un microprocesseur s'apparentent aux instructions du langage Basic mais elles sont plus nombreuses et leur utilisation est plus délicate.

Le langage machine peut permettre d'exécuter les mêmes programmes que le langage Basic, mais l'exécution de ces programmes est bien plus rapide en langage machine.

Lorsqu'un programme écrit en Basic ou en tout autre langage évolué est introduit dans le micro-ordinateur, l'interpréteur ou le compilateur situé en mémoire centrale doit commencer par traduire ce programme en langage machine, le seul compris par le microprocesseur qui contrôle l'ensemble du système informatique.

On comprend alors qu'un programme sera exécuté bien plus rapidement s'il est écrit en langage machine, car ainsi on évite la perte de temps provoquée par la traduction.

Le langage machine permet également d'exécuter des fonctions qui n'ont pas été prévues dans le langage Basic. Dans le cas du micro-ordinateur ZX 81 l'exécution de dessins en haute définition par exemple.

Enfin un programme écrit en langage machine tiendra une place plus réduite dans la mémoire RAM, ce qui est particulièrement appréciable lorsque l'on ne dispose que de 1 Kilo-octet de mémoire vive comme c'est le cas avec la version standard du ZX 81.

MATÉRIEL



CONSTITUTION D'UN MICRO-ORDINATEUR

Tous les ordinateurs, qu'il s'agisse d'un gros ordinateur qui gère un centre de Sécurité Sociale ou d'un micro-ordinateur comme votre ZX 81, sont constitués de la même manière.

Un ordinateur est composé d'un ensemble principal appelé Unité Centrale et de diverses Unités Périphériques. Ces Unités Périphériques sont : les écrans de visualisation, les claviers, les imprimantes, les mémoires de masses (sur support magnétique ou sur cartes perforées), les coupleurs d'entrées et sorties, les alimentations, etc...

L'Unité Centrale de l'ordinateur est la partie la plus importante de l'ordinateur, celle qui contrôle tout le reste et qui effectue tous les traitements et calculs.

Cette Unité Centrale est composée du Processeur et de la Mémoire Centrale, ainsi que des interfaces permettant à ces deux premiers éléments de communiquer avec les Unités Périphériques.

Le Processeur est le cœur de l'ordinateur, la partie active qui contrôle l'ensemble de l'ordinateur et dans laquelle se font toutes les opérations qui sont effectuées par l'ordinateur, que ces opérations soient d'ordre arithmétique (additions...) ou logique (comparaisons).

Le processeur d'un ordinateur peut être de taille réduite, et lorsqu'il est contenu dans un circuit intégré c'est un microprocesseur, mais si la taille d'un microprocesseur est réduite ses performances sont loin de l'être.

Ce qui limite les performances d'un ordinateur est le plus souvent la taille de sa mémoire centrale. Celle-ci est partagée en deux parties. La mémoire permanente ou morte nommée ROM (Read Only Memory) et la mémoire temporaire ou vive nommée RAM (Random Acces Memory).

La mémoire ROM contient les informations qui ne doivent jamais disparaître comme le langage de programmation Basic dans le cas du ZX 81, ainsi que les programmes de gestion des différents éléments de l'ordinateur. Cette mémoire ROM peut seulement être lue.

La mémoire RAM est celle qui contient tous les programmes dont l'utilisateur aura besoin. Le fichier d'affichage fait également partie de la mémoire RAM. Comme la mémoire ROM, la mémoire RAM peut être lue, mais on peut aussi y écrire pour changer les octets contenus dans les cases de cette mémoire ; c'est ce que l'on fait lorsqu'on écrit un programme à l'aide du clavier.

La mémoire RAM a la particularité de perdre les informations qu'elle contient aussitôt que l'on coupe l'alimentation de l'ordinateur, ce qui rend indispensable l'adjonction d'une mémoire de masse pour conserver sur support magnétique les programmes et les autres informations contenues dans la mémoire RAM et éviter qu'elles ne soient perdues à chaque coupure de l'alimentation.

Dans les débuts de l'informatique le processeur était la partie la plus coûteuse des ordinateurs, c'est pourquoi les chercheurs ont eu pour tâche essentielle de réduire le prix et l'encombrement des processeurs d'ordinateurs, sans en altérer les performances.

Le processeur d'un ordinateur occupait plusieurs armoires pleines de matériel électronique ; à mesure que les progrès techniques se poursuivaient la taille et le prix des processeurs se réduisaient. Les tubes électroniques étaient remplacés par des transistors, puis les transistors par des circuits intégrés qui dans un même boîtier réunissaient plusieurs transistors et divers composants.

Le processeur d'un ordinateur peut tenir dans un tiroir, puis sur une seule carte, à mesure que les circuits intégrés se perfectionnaient en intégrant un nombre toujours plus grand de composants.

Finalement on a réussi à faire tenir un processeur complet d'ordinateur dans un seul boîtier de circuit intégré ayant la taille du doigt : le microprocesseur était né.

2 — LES MICROPROCESSEURS

La venue du microprocesseur a marqué le début d'une nouvelle révolution industrielle, les conséquences de celle-ci sont à peine perceptibles actuellement, mais dans l'avenir l'utilisation du microprocesseur va modifier complètement les conditions de travail des individus et la vie de chacun.

Les microprocesseurs permettent en effet de construire non seulement des ordinateurs économiques, mais aussi des machines automatiques capables de travailler seules en assurant automatiquement la production sous une surveillance réduite.

Ces machines automatiques contrôlées par des microprocesseurs coûteront à peine plus cher que des machines équivalentes qui auront besoin pour fonctionner de la présence continuelle d'un ouvrier aux commandes de la machine.

Depuis les débuts du développement de l'informatique on savait qu'un ordinateur pouvait être programmé pour assurer les commandes et les contrôles effectués par des ouvriers. Cela dans des branches aussi diverses que l'industrie, l'agriculture, la comptabilité, le dessin industriel ou la distribution commerciale. Mais à cette époque l'utilisation d'un ordinateur valant plusieurs millions de francs actuels ne pouvait être rentable que dans des conditions exceptionnelles.

La venue de microprocesseurs, en permettent de construire des ordinateurs très économiques, a rendu possible une infinité d'applications que le prix élevé du matériel électronique avait jusqu'à présent freiné.

La programmation en langage machine permet de mieux comprendre ces diverses applications des microprocesseurs.

La puce d'un microprocesseur qui comprend des milliers de transistors est fabriquée à l'aide de techniques qui s'apparentent à celles de la photographie ; ceci permet de réduire d'une manière considérable le prix de revient des microprocesseurs, car on se rend bien compte que la photographie d'un grand nombre de détails, ne sera guère plus coûteuse que celle d'une surface unie.

Une fois qu'un microprocesseur est au point, sa production en grande série ne pose pas de très gros problèmes.

Le microprocesseur Z 80 A qui équipe le micro-ordinateur ZX 81 opère sur des mots de 8 bits ou un octet, chaque bit de l'octet est un chiffre binaire, qui n'a que deux valeurs, 0 ou 1.

La plupart des micro-ordinateurs courants sont aussi équipés de microprocesseurs travaillant également sur des mots de un octet, mais bien souvent ils sont équipés d'un microprocesseur différent du Z 80.

Il existe en effet toute une variété de microprocesseurs, certains travaillent sur des mots de 4 bits, d'autres sur des mots de 16 bits ou même de 32 bits.

Certains microprocesseurs constituent un micro-ordinateur complet dans un seul boîtier de circuit intégré, en incorporant dans ce boîtier en plus du microprocesseur une mémoire ROM et une mémoire RAM. C'est le type de microprocesseur qui se trouve dans des appareils électroménagers perfectionnés.

Le langage machine des divers microprocesseurs est différent, par exemple le langage machine du microprocesseur Z 80 ne peut pas être utilisé avec un microprocesseur 6502 ni avec un microprocesseur 6800.

Par contre, une fois que l'on sait programmer un microprocesseur en langage machine il n'est guère difficile de transposer les connaissances acquises pour programmer un autre microprocesseur. Tout comme lorsqu'on sait conduire un véhicule automobile, il n'est guère difficile de transposer les connaissances acquises pour conduire un autre véhicule même très différent.

Un microprocesseur fonctionne tout comme le processeur d'un gros ordinateur en répétant constamment le cycle suivant à la cadence de son oscillateur d'horloge :

1. **Lecture de l'adresse de la case mémoire contenant la prochaine instruction à exécuter.**

2. **Recherche dans la mémoire centrale (ROM et RAM) du mot binaire se trouvant à l'adresse indiquée.**

3. **Lecture de l'octet se trouvant à cette adresse par exemple une instruction.**

4. **Exécution de l'instruction se trouvant à l'adresse indiquée (calcul arithmétique, opération logique, saut du programme à une autre adresse, etc.).**

5. Implantation dans le registre d'adresse du microprocesseur de l'adresse de la case mémoire qui contient la prochaine instruction à exécuter.

Ensuite le cycle recommence à son début mais certaines instructions comportant plusieurs octets nécessitent plusieurs cycles avant d'être complétées.

3 — LA NUMERATION BINAIRE

Tout comme le processeur des plus gros ordinateurs, le microprocesseur Z 80 A qui contrôle le micro-ordinateur ZX 81 ne connaît qu'une seule sorte de numération, la *numération binaire,* et c'est dans cette numération que les instructions du langage machine sont placées dans les cases de la mémoire RAM.

La numération binaire ne connaît que deux chiffres **0** ou **1**, alors qu'en numération décimale que nous utilisons habituellement on connaît les 10 chiffres : 0 à 9.

En informatique le chiffre 0 est en général marqué par l'absence de tension sur un des conducteurs qui transmet les données ou les adresses entre le microprocesseur et ses périphériques, et le chiffre 1 se traduit par la présence de tension. Un nombre binaire est formé d'une succession de chiffres binaires 0 et 1 tout comme un nombre décimal est formé par une succession de chiffres décimaux de 0 à 9. En jargon informatique, un chiffre binaire s'appelle bit.

Le micro-ordinateur ZX 81 utilise la numération binaire pour effectuer toutes les opérations qui sont programmées par l'utilisateur.

En employant le langage Basic on ne s'aperçoit pas de l'utilisation de la numération décimale et on reçoit les résultats de la même façon. Mais en réalité, entre temps, le microprocesseur a transformé les informations décimales qu'il a reçues en informations binaires avant de les placer dans les cases de la mémoire RAM.

Pour quelqu'un qui n'est pas habitué à la numération binaire celle-ci peut sembler compliquée, il n'en est rien et les ordinateurs qui effectuent toutes les opérations que nous leur soumettons en numération binaire, le font avec aisance et rapidité.

Lorsque nous programmerons notre ZX 81 en langage machine nous ne pourrons plus compter sur le langage Basic pour effectuer la traduction en binaire. Mais comme l'écriture d'un nombre binaire comportant un grand nombre de zéro et de un entraîne facilement des erreurs, nous pourrons utiliser la numération hexadécimale. Le microprocesseur transforme facilement un octet binaire en nombre hexadécimal ou inversement, un nombre hexadécimal en binaire.

La liste suivante montre les 20 premiers nombres entiers dans les trois types de numération employés couramment en informatique : décimale, hexadécimale et binaire.

Les nombres hexadécimaux et binaires sont regroupés sous la forme d'un octet, et c'est sous cette forme que les nombres binaires vont se retrouver dans les cases de la mémoire RAM ou dans les registres du microprocesseur.

Numération Décimale	**Numération Hexadécimale**	**Numération Binaire**
0	00	00000000
1	01	00000001
2	02	00000010
3	03	00000011
4	04	00000100
5	05	00000101
6	06	00000110
7	07	00000111
8	08	00001000
9	09	00001001
10	0A	00001010
11	0B	00001011
12	0C	00001100
13	0D	00001101
14	0E	00001110
15	0F	00001111
16	10	00010000
17	11	00010001
18	12	00010010
19	13	00010011
20	14	00010100

Un octet binaire peut avoir une valeur quelconque entre la valeur minimale zéro et la valeur maximale 255.

Lorsqu'un octet est placé dans une case de la mémoire ou dans un registre du microprocesseur, zéro se présente sous la forme binaire 00000000 et 255 sous la forme 11111111.

Dans un octet binaire la valeur d'un bit dépend de la place que ce bit occupe dans l'octet, tout comme dans un nombre décimal la valeur d'un chiffre dépend de la place de ce chiffre dans le nombre, suivant qu'il est placé au rang des unités, des dizaines ou des centaines.

Le tableau suivant indique la valeur décimale et hexadécimale de chacun des bits d'un octet.

Bit	7	6	5	4	3	2	1	0
Valeur Décimale	128	64	32	16	8	4	2	1
Valeur Hexadécimale	80	40	20	10	8	4	2	1

Pour connaître la valeur d'un octet binaire on additionne la valeur de chacun des bits de cet octet qui est à 1, les bits à zéro étant simplement ignorés.

Ainsi la valeur décimale de l'octet binaire 01001000 dont les bits 3 et 6 sont à 1 est de 64 + 8 soit 72 (les bits sont numérotés 0,1...7 de droite à gauche).

La valeur hexadécimale de ce même octet est 40 + 8 soit 48 ce qui représente la même valeur.

Un octet dont tous les bits sont à 1, a la valeur 255 en numération décimale et FF en numération hexadécimale.

Nous avons vu qu'un octet pouvait représenter un nombre décimal de 0 à 255, mais il peut représenterd'autres valeurs par exemple une lettre ou un caractère graphique ; il peut aussi représenter un nombre négatif ou positif de – 127 à + 127.

A l'aide d'un convertisseur analogique-digital un ordinateur peut lire une grandeur électrique et dans ce cas un octet peut représenter une tension électrique entre 0 et 255 Volts par exemple. On peut ainsi établir un pro-

gramme pour réguler la tension d'un alternateur à 220 Volts à + ou − 2 Volts près.

Il est possible d'établir des programmes en langage machine pour qu'un octet effectue des commandes automatiques : un octet peut ainsi commander directement 8 appareils électriques différents (Moteurs, Vannes électriques etc...). En utilisant des décodeurs qui sont des circuits intégrés courants, on peut aller jusqu'à la commande de 255 appareils électriques différents à partir d'informations contenues dans un seul octet.

4 — LE MICROPROCESSEUR Z 80

Le microprocesseur Z 80 qui équipe le micro-ordinateur ZX 81 est dérivé du microprocesseur 8080 qui a connu une grande diffusion il y a quelques années.

Le microprocesseur Z 80 peut comprendre le langage machine du 8080. Ainsi un programme écrit pour un microprocesseur 8080 fonctionnera également sur un microprocesseur Z 80. Cependant la réciproque n'est pas vraie.

En effet le microprocesseur Z 80 comporte un grand nombre d'instructions que le microprocesseur 8080 ne peut pas comprendre.

Le microprocesseur 8080 comporte 4 500 transistors et a un jeu de 78 instructions, alors que le microprocesseur Z 80 qui se présente dans le même boîtier avec 40 broches comprend 8 000 transistors et son jeu comporte 158 instructions (696 instructions si l'on tient compte des différents modes d'adressage).

La version standard du Z 80 travaille avec une fréquence d'horloge de 2,5 MHz mais il existe une version plus rapide le Z 80 A qui travaille avec une fréquence d'horloge de 4 MHz c'est cette dernière version qui équipe le micro-ordinateur ZX 81. Naturellement les deux versions ont le même langage machine.

Les registres d'un microprocesseur sont très importants pour la programmation en langage machine. Le microprocesseur Z 80 possède un total de 22 registres accessibles à l'utilisateur :

— 2 registres accumulateurs A et A'.

— 2 registres de mots d'état PSW et PSW'.

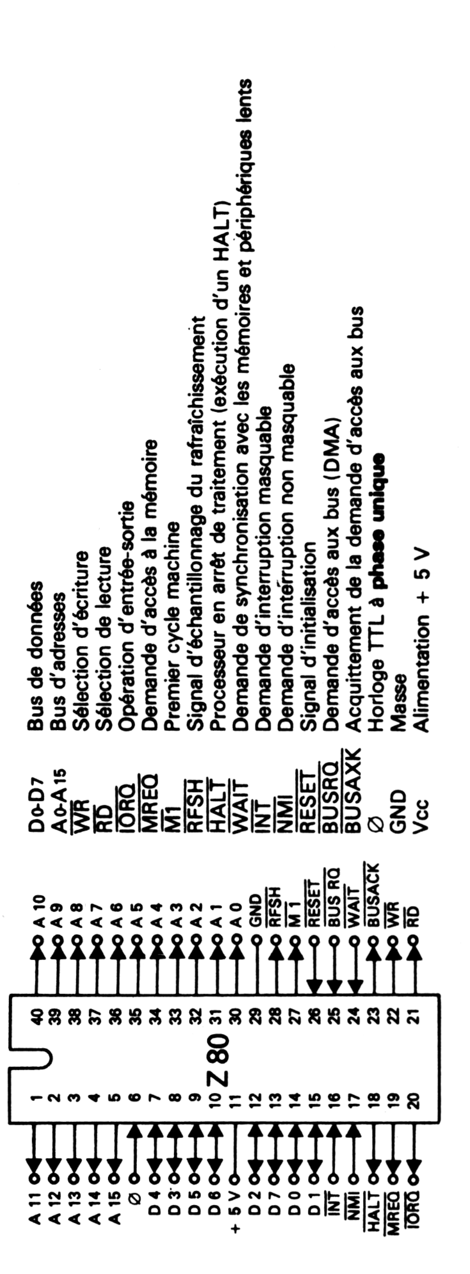

Z 80
A 11, A 12, A 13, A 14, A 15, Ø, D 4, D 3, D 5, D 6, + 5 V, D 2, D 7, D 0, D 1, INT, NMI, HALT, MREQ, IORQ
A 10, A 9, A 8, A 7, A 6, A 5, A 4, A 3, A 2, A 1, A 0, GND, RFSH, M 1, RESET, BUS RQ, WAIT, BUSACK, WR, RD
D0-D7 Bus de données
A0-A15 Bus d'adresses
WR Sélection d'écriture
RD Sélection de lecture
IORQ Opération d'entrée-sortie
MREQ Demande d'accès à la mémoire
M1 Premier cycle machine
RFSH Signal d'échantillonnage du rafraîchissement
HALT Processeur en arrêt de traitement (exécution d'un HALT)
WAIT Demande de synchronisation avec les mémoires et périphériques lents
INT Demande d'interruption masquable
NMI Demande d'interruption non masquable
RESET Signal d'initialisation
BUSRQ Demande d'accès aux bus (DMA)
BUSAXK Acquittement de la demande d'accès aux bus
Ø Horloge TTL à phase unique
GND Masse
Vcc Alimentation + 5 V

Figure 1

— 2 jeux de 6 registres généraux sur 8 bits qui peuvent se combiner pour donner 2 jeux de 3 registres 16 bits.

HL et H'L'
BC et B'C'
DE et D'E'

— Un compteur ordinal 16 bits PC.

— Un compteur de rafraîchissement R.

— Un registre de pages d'adresse I (utilisé pour les interruptions).

— Un registre pointeur de pile 16 bits SP.

— 2 registres d'index 16 bits IX et IY.,

La figure 1 (voir page précédente) montre le microprocesseur Z 80 ainsi que son brochage. La figure 2 montre les registres du Z 80.

On remarque que ces registres forment deux jeux symétriques, mais un seul jeu est disponible à la fois et c'est sur celui-ci que travaille le programmeur. Il peut passer sur l'autre jeu en utilisant une instruction d'échange.

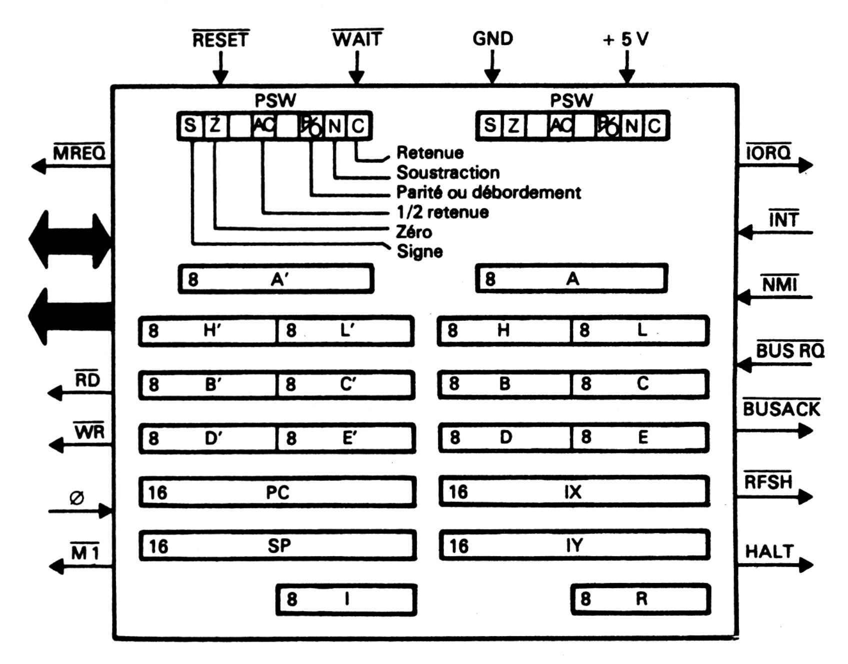


Figure 2

LES INSTRUCTIONS DU LANGAGE MACHINE

Nous allons voir la liste des instructions du langage machine servant à programmer le microprocesseur Z 80. Ces instructions sont présentées sous leur forme mnémonique qui est celle adoptée pour le langage assembleur du Z 80.

Le code mnémonique des instructions d'un microprocesseur est une abréviation de la désignation de l'instruction en langue anglaise, par exemple l'instruction LD est l'abréviation de LOAD qui donne l'ordre au microprocesseur de transférer un octet.

Lorsqu'on utilise le langage machine c'est le code opératoire correspondant à cette instruction qui est utilisé.

La liste des codes opératoires est donnée en annexe de cet ouvrage ; on trouve également cette liste dans l'annexe A du manuel de programmation SINCLAIR livré avec le micro-ordinateur ZX 81.

Par exemple, nous voulons employer l'instruction DEC A qui donne l'ordre au microprocesseur de décrémenter d'une unité l'octet contenu dans le registre A ; cette instruction ayant pour code hexadécimal 3D, c'est donc ce code 3D que nous placerons dans une case de la mémoire RAM lorsque nous programmerons en langage machine.

Pour chaque code mnémonique nous indiquerons un exemple de code hexadécimal.

ADC Addition avec retenue ; un octet contenu dans un registre ou à un emplacement de la mémoire est additionné avec le contenu du registre accumulateur A ; le résultat de l'opération se retrouve dans A.
Exemple : 88 - Addition des registres A et B.

ADD Ressemble à ADC mais la retenue n'est pas utilisée dans le calcul initial, elle est seulement utilisée lors du résultat final.
Exemple : 80 - Addition des registres A et B.

AND ET Logique. L'octet contenu dans le registre A subit un ET Logique avec un autre octet, le résultat de AND, 00 est toujours zéro et AND, FF laisse A inchangé.
Exemple : A1 - ET Logique entre A et C.

BIT Permet de contrôler si un bit de l'octet contenu dans un registre est à 0 ou à 1.
Exemple : CB 68 contrôle la valeur du bit 5 de B.

CALL Ressemble au GOSUB du Basic ; cette instruction provoque un saut de programme à l'adresse indiquée à la suite de CALL et revient à l'adresse d'origine avec l'instruction RET.
Exemple : CD 0808 le programme saute à l'adresse Hexa 0808.

CCF « *Complement Carry Flag* » si le bit de retenue est 1 il passe à 0, s'il est 0 il passe à 1.
Exemple : 3F ; (c'est d'ailleurs la seule possibilité).

CP Ordre de comparaison,.la comparaison de deux registres peut faire l'objet d'un saut de programme si le contenu des deux est égal.
Exemple : BC compare les registres A et H.

CPD Comparaison avec décrément semblable à CP mais après la comparaison le registre comparé à A est décrémenté d'une unité.
Exemple : ED A9 comparaison de HL avec A.

CPDR Semblable à CPD mais l'instruction est répétée continuellement jusqu'à ce que l'égalité avec A soit obtenue ou que le registre arrive à zéro.
Exemple : ED 89 comparaison de HL avec A.

CPI Semblable à CPD mais le registre est incrémenté au lieu d'être décrémenté.
Exemple : ED A1 comparaison de HL avec A.

CPIR Semblable à CPDR mais le registre est incrémenté.
Exemple : ED B1 comparaison de HL avec A.

CPL « *Complement* » chacun des bits du registre A est inversé.
Exemple : 2F.

DAA « *Decimal Adjust Accumulator* » change le contenu de l'accumulateur en décimal. Si le contenu de l'accumulateur est de 2A hexadécimal, il devient 42 décimal.
Exemple : 27.

DEC Décrémentation d'un registre.
Exemple : 3D décrémentation de A.

DI « *Disable Interrupt* ». Masque les interruptions du microprocesseur.
Exemple : F3.

DJNZ « *Decrement and Jump if Not Zero* ». Décrémente le registre B et saute le nombre de pas indiqué si le registre B ne contient pas zéro.
Exemple : 10,20.

EI « *Enable Interrupt* ». Permet les interruptions.
Exemple : FB.

EX Echange le contenu de deux registres.
Exemple : EB échange DE et HL.

EXX Echange l'ensemble des registres symétriques BC et B'C' etc...
Exemple : D9.

HALT Bloque le microprocesseur.
Exemple : 76.

IM « *Interrupt Mode* ». Sélectionne le mode d'interruption du microprocesseur.
Exemple : ED 56 (Mode 1).

IN « *Input* ». Entrée en série permet au microprocesseur de charger, par exemple, un programme enregistré sur cassette.
Exemple : DB 20.

INC Ordre d'incrémentation, l'exemple qui suit incrémente le registre A.
Exemple : 3C.

IND « *Input with Decrement* » semblable à IN mais avec un décrément.
Exemple : ED AA.

INDR Semblable à IND mais l'instruction est répétée continuellement.
Exemple : ED BA.

INI Semblable à IND mais le registre est incrémenté au lieu d'être décrémenté.
Exemple : ED A2.

INIR Semblable à INI mais l'instruction est répétée continuellement.
Exemple : ED B2.

JP Ordre de saut de programme, cet ordre est semblable au GOTO en Basic mais la destination est une adresse de la mémoire centrale au lieu d'être un numéro de ligne. Existe avec de nombreuses variantes ; l'exemple qui suit donne l'ordre de sauter à l'adresse indiquée par la paire de registres HL.
Exemple : E9.

JR Ordre de saut relatif ; peut exister sous forme conditionnelle JRZ, JRNZ. L'exemple qui suit donne l'ordre de sauter 20 cases mémoires plus loin en hexadécimal (32 en décimal).
Exemple : 18 20.

LD Instruction de chargement, sert à transférer les octets, c'est l'instruction la plus utilisée du langage machine elle comporte de nombreuses variantes que nous verrons lorsque nous programmerons en langage machine ; l'exemple qui suit charge l'octet 17 dans le registre A.
Exemple : 3E 17.

LDD Chargement avec décrément.
Exemple : ED A8.

LDDR Semblable à LDD mais l'instruction est répétée continuellement jusqu'à ce que le zéro soit atteint.
Exemple : ED B8.

LDI Semblable à LDD mais le registre est incrémenté au lieu d'être décrémenté.
Exemple : ED A0.

LDIR Semblable à LDDR mais l'octet est incrémenté au lieu d'être décrémenté.
Exemple : ED B0.

NEG Cette instruction modifie l'octet contenu dans l'accumulateur qui est inversé ainsi si cet octet est 06 il devient FA soit −6.
Exemple : ED 44.

NOP « *NO OPERATION* ». Cette instruction est utile bien qu'elle ne fasse rien ; elle permet des temporisations ou encore de réserver des cases mémoires dans un programme.
Exemple : 00.

OR L'octet subit un OU logique dans un registre. L'exemple suivant montre OR,A : si l'octet contenu dans A est 00 OR,A équivaut à LD,A, si A contient FF, OR est sans effet.
Exemple : B7.

OUT Même type d'instruction que IN pour transférer en série le contenu d'un registre vers un périphérique extérieur ; permet par exemple de sauver un programme sur une cassette.
Exemple : ED 79.

OUTD « *Output With Decrement* ». Semblable à OUT mais en plus décrémentation.
Exemple : ED AB.

OTDR Semblable à OUTD mais répétition de l'instruction jusqu'à ce que l'on arrive à 0.
Exemple : ED BB.

OUTI « *Output With Increment* ». Semblable à OUTD mais avec incrémentation.
Exemple : ED A3.

OTIR Semblable à OTDR avec incrémentation au lieu de décrémentation.
Exemple : ED B3.

POP Enlève 2 octets de la pile pour les charger dans une paire de registres, HL dans l'exemple suivant.
Exemple : E1.

PUSH L'opposé de POP met les 2 octets contenus dans une paire de registres dans la pile, HL dans l'exemple.
Exemple : E5.

RES « *Reset* ». Met un bit à zéro dans un registre.
Exemple : CB 8F.

RET Ordre de retour lors d'un sous-programme, c'est le complément de CALL.
Exemple : C9.

RETI Retour d'interruption, met fin à une interruption.
Exemple : ED 4D.

RETN Semblable à RETI utilisé pour mettre fin aux interruptions non masquables.
Exemple : ED 45.

RL Rotation vers la gauche du contenu d'un registre.
Exemple : CB 13.

RLA « *Rotate Left Accumulator* ». Rotation vers la gauche du contenu de l'accumulateur.
Exemple : CB 17.

RLC Semblable à RLA pour registre quelconque, le registre D dans l'exemple suivant.
Exemple : CB 02.

RLCA Semblable à RLA mais le bit le plus à gauche de l'octet va simultanément dans la case de droite du registre et dans la case de retenue.
Exemple : 07.

RLD « *Rotate Left Decimal* ». Rotation vers la gauche de 4 bits entre A et HL.
Exemple : ED 6F.

RR Semblable à RL mais avec rotation vers la droite du contenu d'un registre, B dans l'exemple.
Exemple : CB 18.

RRA Semblable à RLA mais les bits de l'octet sont déplacés vers la droite.
Exemple : 1F.

RRC Semblable à RLC mais avec rotation vers la droite du contenu d'un registre, le registre B dans l'exemple.
Exemple : CB 08.

RRCA Semblable à RLCA mais avec rotation vers la droite.
Exemple : 0F.

RRD Semblable à RLD mais la rotation est vers la droite.
Exemple : ED 67.

RST Initialise le microprocesseur, l'exemple choisi RST,0 a le même effet qu'une coupure momentanée de l'alimentation.
Exemple : C7.

SBC Soustrait de A le contenu d'un registre en tenant compte du bit de retenue, le contenu du registre A est le résultat de l'opération.
Exemple : 9A.

SET Met à 1 l'un des bits d'un registre.
Exemple : CB CF.

SLA « *Shift Left Arithmetic* ». Décalage arithmétique à gauche semblable à RL sauf que le bit de droite passe à zéro, l'exemple porte sur le registre B.
Exemple : CB 20.

SRA « *Shift Right Arithmetic* ». Décalage arithmétique à droite, le bit 6 du registre prend la valeur du bit 7 ; cette commande équivaut à diviser l'octet contenu dans le registre par 2, l'exemple porte sur le registre B.
Exemple : CB 28.

SRL Décalage logique à droite, l'exemple porte sur B.
Exemple : CB 38.

SUB Soustrait de l'octet contenu dans A, l'octet contenu dans un registre, l'exemple porte sur B.
Exemple : 90.

XOR OU logique exclusif entre l'accumulateur A et un registre, l'exemple choisi XOR, A met à zéro le bit de retenue et le contenu de l'accumulateur.
Exemple : AF.

Présentées ainsi que nous venons de le voir, les instructions du Z 80 peuvent sembler difficiles, mais nous verrons que tout paraîtra plus simple lorsque dans les premiers programmes en langage machine, nous appliquerons ces mêmes instructions à des exemples pratiques.

Nous retrouverons ces instructions avec leur code opératoire hexadécimal en annexe de cet ouvrage.

5 — ORGANISATION DE LA MÉMOIRE DU MICRO-ORDINATEUR ZX 81

L'emploi du langage machine pour programmer un micro-ordinateur nécessite une certaine connaissance de la mémoire centrale de celui-ci.

Le manuel SINCLAIR livré avec ce micro-ordinateur contient un grand nombre d'informations utiles pour la programmation (chapitres 27 et 28 de ce manuel), mais certaines informations complémentaires sont nécessaires pour mieux comprendre l'organisation de la mémoire centrale de ce micro-ordinateur et son utilisation avec les programmes en langage machine.

Lorsqu'on programme un micro-ordinateur en Basic, le programme se place automatiquement dans la mémoire RAM sans que le programmeur ait à se préoccuper des emplacements mémoire où va se loger le programme.

Il n'en est plus de même lorsque l'on programme en langage machine et dans ce cas le programmeur doit tenir compte des zones de mémoire où il place son programme, et ses données.

La mémoire ROM du micro-ordinateur ZX 81 qui contient le langage Basic débute à l'adresse 0 et va jusqu'à l'adresse 8191. En numération hexadécimale elle va de 0000 à 1FFF.

Les adresses de la mémoire de 8192 à 16383, en numération hexadécimale de 2000 à 3FFF, reproduisent une copie de la mémoire ROM. Ces adresses peuvent servir pour y placer des éléments de mémoire RAM supplémentaires.

La mémoire RAM normale du micro-ordinateur ZX 81 débute à l'adresse 16384 (4000 en hexadécimal) et va jusqu'à l'adresse 17407 (43FF en hexadécimal) dans le cas de la version 1 K du ZX 81 standard.

Lorsqu'on place le module d'extension mémoire de 16 K sur le micro-ordinateur ZX 81 la mémoire normale RAM du ZX 81 va de l'adresse 16 384 à l'adresse 32767 (de 4000 à 7FFF en hexadécimal).

C'est dans la mémoire RAM que nous allons implanter nos programmes en langage machine. Cependant la totalité de cette mémoire RAM n'est pas disponible : les chapitres 27 et 28 du manuel SINCLAIR sur le ZX 81 indiquent que les adresses 16384 à 16508 sont affectées à diverses fonctions.

Aussi lorsque nous écrivons un programme en langage Basic celui-ci va se placer à partir de la première adresse disponible de la mémoire, c'est-à-dire l'adresse 16509.

Les 5 premiers octets d'un programme contiennent des renseignements destinés à l'interpréteur Basic, même si on place ensuite un programme en langage machine, si bien que le premier octet du langage machine sera placé normalement à l'adresse 16514 comme nous le verrons plus loin.

Bien sûr lorsque nous implanterons un programme en langage machine dans la mémoire RAM nous pourrons placer ce programme à partir d'une adresse quelconque, par exemple l'adresse 17000, à condition que cette partie de la mémoire soit disponible.

Mais si nous voulons pouvoir sauver sur cassette nos programmes en langage machine, le choix devient plus restreint et pratiquement nous sommes conduits à commencer nos programmes en langage machine à l'adresse 16514 dans une instruction REM.

Le micro-ordinateur ZX 81 dispose de trois instructions Basic permettant le lien avec des programmes en langage machine : ce sont PEEK, POKE et USR.

PEEK lit le contenu d'une case mémoire.

POKE place un octet dans une case mémoire.

USR permet de passer du Basic au langage machine, ainsi USR 17000 donne l'ordre de passer au programme machine débutant à l'adresse 17000.

Observons en Basic la mémoire centrale du ZX 81

Pour commencer, nous allons voir quelques petits programmes, écrits en langage Basic, qui vont nous permettre de mieux comprendre le fonctionnement de la mémoire centrale du ZX 81.

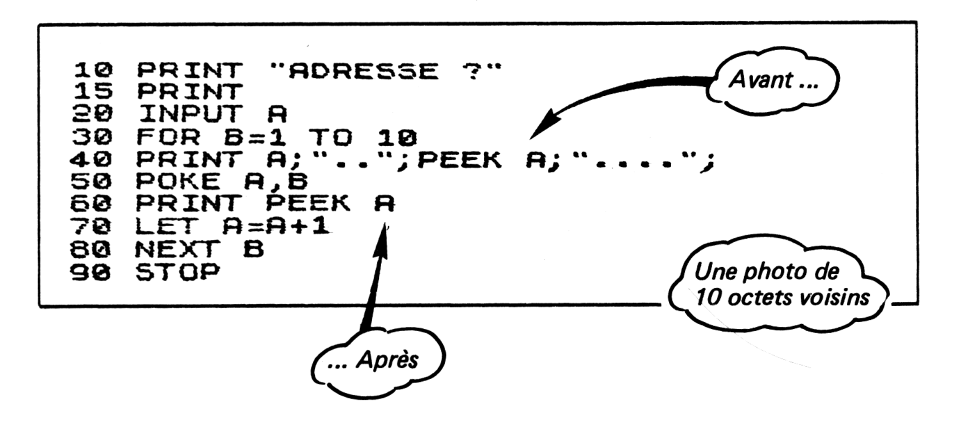
A l'aide du clavier si nous écrivons PRINT PEEK 100, une fois la touche NEW LINE pressée nous verrons s'afficher 20 sur l'écran du téléviseur. Ce nombre correspond à l'octet contenu dans la case mémoire à l'adresse 100, et comme cette adresse correspond à une case de la mémoire ROM son contenu sera toujours le même.

On peut procéder de la même manière pour examiner les octets contenus à d'autres adresses de la mémoire ROM ou RAM du ZX 81.



EXAMEN DE LA MÉMOIRE

Le programme qui suit va nous laisser choisir une adresse de la mémoire centrale et examiner l'octet contenu à cette adresse et aux neuf adresses suivantes. On pourra alors modifier le contenu de ces 10 cases mémoire en inscrivant les nombres de 1 à 10. Nous examinerons alors à nouveau leur contenu.




```
10 PRINT "ADRESSE ?"
15 PRINT
20 INPUT A
30 FOR B=1 TO 10
40 PRINT A;"..";PEEK A;"....";
50 POKE A,B
60 PRINT PEEK A
70 LET A=A+1
80 NEXT B
90 STOP
```

Voyons trois exemples d'application de ce programme.

Dans le premier exemple l'adresse choisie est 100. Comme cette adresse et les suivantes correspondent à des cases de la mémoire ROM les octets de ces cases mémoire restent inchangés.

```
ADRESSE ?

100..20....20
101..255....255
102..8....8
103..60....60
104..250....250
105..109....109
106..0....0
107..40....40
108..2....2
109..8....8
```

On ne change pas le contenu des mémoires mortes

Dans le deuxième exemple, l'adresse choisie est 17000, et puisqu'elle se trouve dans la mémoire RAM, les cases examinées ont leur contenu modifié par le programme.

```
ADRESSE ?

17000..0....1
17001..0....2
17002..0....3
17003..0....4
17004..0....5
17005..0....6
17006..0....7
17007..0....8
17008..0....9
17009..0....10
```

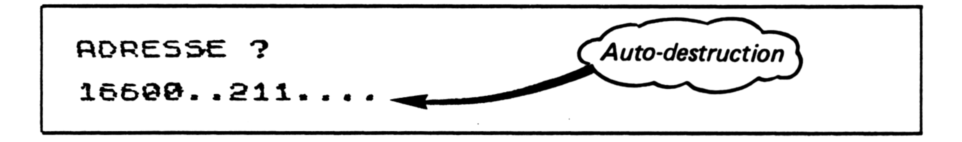
Il n'y avait rien, on a placé 1,..., 10

Dans le troisième exemple, l'adresse de départ est 16600.

C'est une adresse située dans la mémoire RAM, mais elle correspond à des cases mémoire où le programme lui-même est placé.

Cet exemple est surtout destiné à nous montrer que l'instruction POKE peut détruire un programme et qu'elle doit être employée avec précaution.

Ce programme se bloque dès le départ lorsque l'adresse 16600 est inscrite et que le micro-ordinateur ZX 81 cherche à exécuter un POKE à cette adresse.



Lorsqu'on regarde à nouveau le programme on s'aperçoit que celui-ci a été altéré. A la ligne 60, PEEK a disparu pour être remplacé par un carré noir. Cette altération a été provoquée par le POKE concernant l'adresse 16600. Voici le texte du programme altéré :

```
10 PRINT "ADRESSE ?"
15 PRINT
20 INPUT A
30 FOR B=1 TO 10
40 PRINT A;"..";PEEK A;"....";
50 POKE A,B
60 PRINT ■A
70 LET A=A+1
80 NEXT B
90 STOP
```

Cette ligne n'est plus ce qu'elle était !



MODIFICATION AUTOMATIQUE D'UN PROGRAMME (1 K)

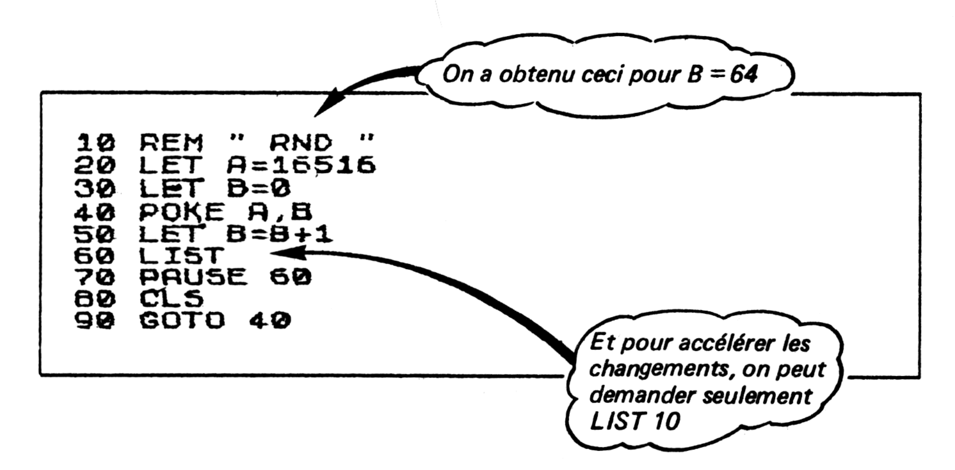
Nous allons voir un autre programme, qui nous montrera comment une instruction POKE dans la zone mémoire contenant le programme peut le modifier. Ce genre de modification va bientôt nous rendre service !

La ligne 10 de ce programme comprend une chaîne de caractères composée de 3 espaces ; c'est à l'emplacement du 2ᵉ espace que l'instruction POKE de la ligne 40 va placer successivement les nombres allant de zéro à... qui seront modifiés à chaque pause du programme.

3 espaces

```
10 REM "   "
20 LET A=16516
30 LET B=0
40 POKE A,B
50 LET B=B+1
60 LIST
70 PAUSE 60
80 CLS
90 GOTO 40
```

L'exemple suivant montre le programme dont la ligne 10 est modifiée automatiquement d'une manière constante.




```
10 REM " RND "
20 LET A=16516
30 LET B=0
40 POKE A,B
50 LET B=B+1
60 LIST
70 PAUSE 60
80 CLS
90 GOTO 40
```

LE LANGAGE MACHINE

1 — ÉCRITURE DIRECTE DANS LE FICHIER D'AFFICHAGE

Avant d'aborder le langage machine il est intéressant de voir une autre application des instructions PEEK et POKE : c'est l'écriture directe dans le fichier d'affichage du micro-ordinateur ZX 81.

Le fichier d'affichage fait partie de la mémoire RAM, c'est le contenu de ce fichier que nous voyons s'afficher sur l'écran.

L'emplacement du fichier d'affichage dans la mémoire RAM est variable et est dépendante du programme en cours.

Lorsque le ZX 81 ne dispose que de 1 K de mémoire RAM, ce fichier est réduit au minimum, il peut être réduit à 25 octets correspondants à des NEW LINE, soit 25 lignes vides sur l'écran.

Par contre lorsqu'on utilise le module d'extension mémoire qui porte à 16 K la mémoire RAM du ZX 81 la totalité du fichier d'affichage reste disponible, et nous pouvons écrire directement dans ce fichier d'affichage à l'aide de POKE, sans passer par les instructions PRINT. Tout ce qui sera écrit dans le fichier d'affichage apparaîtra sur l'écran.

Cette méthode permet d'afficher sur l'écran des mouvements plus rapides que ne le permettrait l'utilisation des instructions PRINT, ce qui sera très utile dans bien des programmes de jeux.

Normalement les programmes basés sur l'écriture directe dans le fichier d'affichage nécessitent l'utilisation du module 16 K d'extension mémoire, même si ces programmes sont courts comme le programme 3 que nous allons voir.

3 VA ET VIENT (16 K)

Ce programme montre le va et vient d'une étoile entre deux carrés noirs dans le haut de l'écran.

Ce programme début à la ligne 10 par la recherche de l'adresse du début du fichier d'affichage ; cette adresse est contenue dans les cases de la mémoire RAM aux adresses 16398 et 16399.

Les instructions PEEK des lignes 90 et 130 testent une adresse du fichier d'affichage pour déterminer le sens du déplacement de l'étoile : si la case située à droite de l'étoile est vide, on efface l'étoile et on la fait progresser d'une case vers la droite, si non on repart à gauche jusqu'à ce que l'étoile touche le carré de gauche. Et ainsi de suite... .

```
  10 LET A=PEEK 16398+256*PEEK 1
6399+66
  20 POKE A,128
  30 POKE A+31,128
  40 LET A=A+1
  50 POKE A,0
  60 LET A=A+1
  80 POKE A,23
  90 IF PEEK (A+1)=0 THEN GOTO 5
0
 100 POKE A,0
 110 LET A=A-1
 120 POKE A,23
 130 IF PEEK (A-1)=0 THEN GOTO 1
00
 140 GOTO 50
```

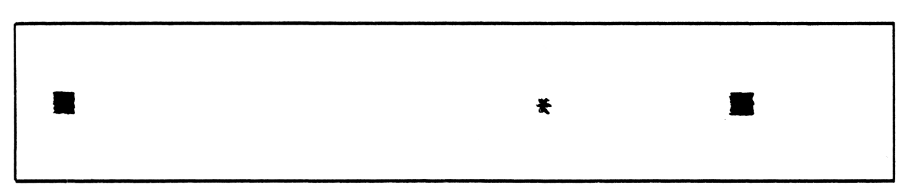
Adresse du caractère situé en haut, à gauche sur l'écran

Les deux carrés

Effacement de l'étoile

Déplacement vers la droite

L'exemple suivant représente une image obtenue grâce à ce programme : on voit ainsi l'étoile parcourir l'écran en un va et vient continuel entre les deux carrés noirs.



Le même principe permet de programmer une infinité de jeux interactifs sur l'écran de votre téléviseur.

4 SQUASH (16 K)

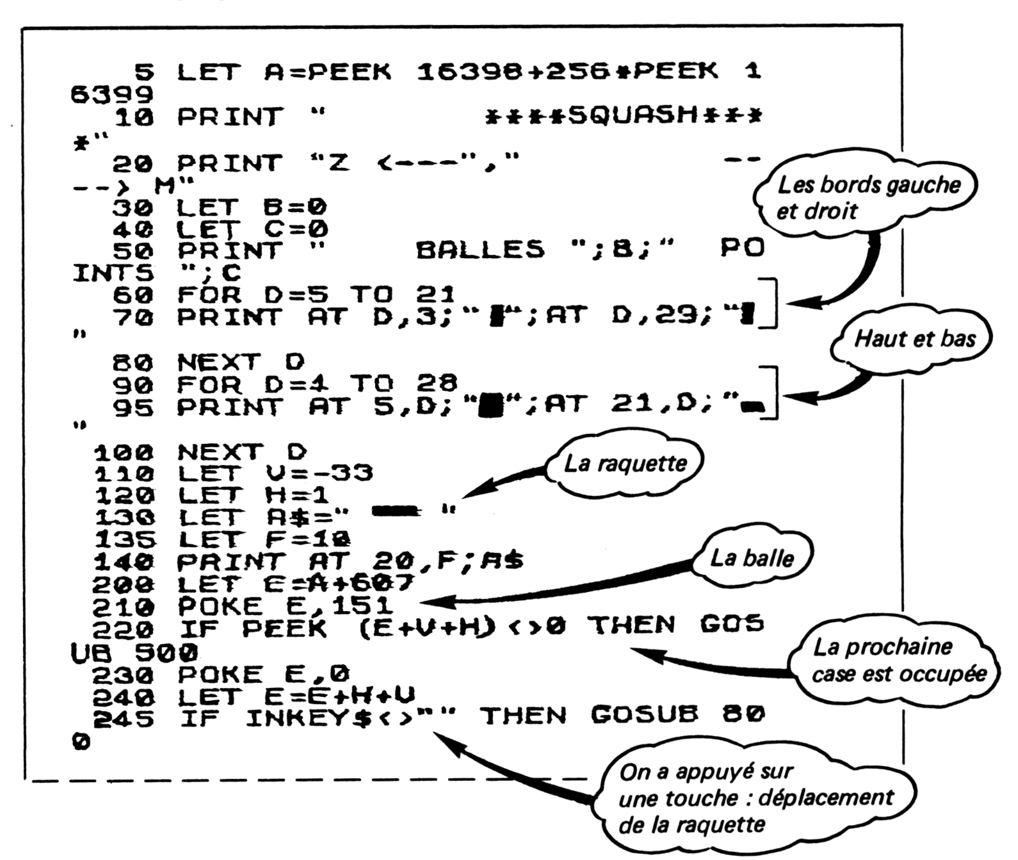
A titre d'exemple, voici le programme d'un jeu de Squash.

Dans ce jeu vous devez rattraper une balle à l'aide d'une raquette. Chaque fois que vous rattrapez la balle vous marquez un point. Le jeu se termine lorsque vous avez laissé passer 15 balles.

La touche Z du clavier dirige la raquette vers la gauche et la touche M vers la droite.

Les mouvements de la balle sont provoqués par les instructions PEEK et POKE de la même manière que dans le programme précédent.

Voici ce programme : essayez de marquer le maximum de points.

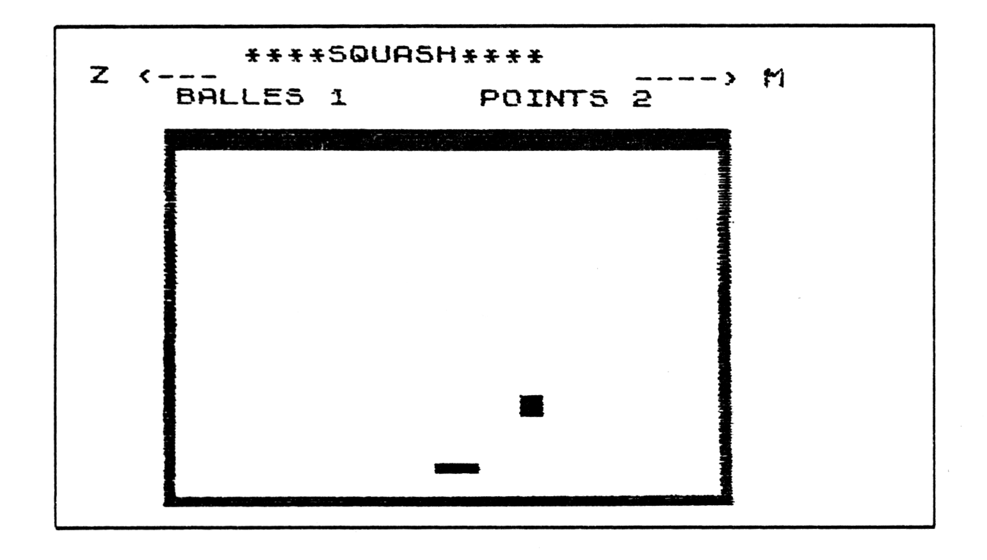


```
   5 LET A=PEEK 16398+256*PEEK 1
6399
  10 PRINT "         ****SQUASH***
*"
  20 PRINT "Z <---","          --
--> M"
  30 LET B=0
  40 LET C=0
  50 PRINT "    BALLES ";B;"  PO
INTS ";C
  60 FOR D=5 TO 21
  70 PRINT AT D,3;"█";AT D,29;"█
"
  80 NEXT D
  90 FOR D=4 TO 28
  95 PRINT AT 5,D;"█";AT 21,D;"▄
"
 100 NEXT D
 110 LET V=-33
 120 LET H=1
 130 LET A$=" ▀▀▀ "
 135 LET F=10
 140 PRINT AT 20,F;A$
 200 LET E=A+607
 210 POKE E,151
 220 IF PEEK (E+V+H)<>0 THEN GOS
UB 500
 230 POKE E,0
 240 LET E=E+H+V
 245 IF INKEY$<>"" THEN GOSUB 80
0
```

```
 250 GOTO 210
 300 REM PIERRE SIRVEN
 500 IF PEEK (E+V+H)=128 THEN LE
T V=ABS V
 510 IF PEEK (E+V+H)=5 THEN LET
H=-H
 520 IF PEEK (E+V+H)=133 THEN LE
T H=ABS H
 530 IF PEEK (E+V+H)=131 THEN GO
SUB 600
 540 IF PEEK (E+V+H)=0 THEN GOSU
B 700
 550 RETURN
 600 LET V=-V
 610 LET B=B+1
 620 PRINT AT 2,11;B
 630 IF B>=15 THEN GOTO 900
 640 RETURN
 700 LET V=-V
 710 LET C=C+1
 720 PRINT AT 2,25;C
 730 RETURN
 800 LET F=F+(INKEY$="M")-(INKEY
$="Z")
 810 IF F>=26 THEN LET F=F-1
 820 IF F<=3 THEN LET F=F+1
 840 PRINT AT 20,F;A$
 850 RETURN
 900 PAUSE 600
 910 CLS
 920 GOTO 5
```

Bord droit (500) — *Bord haut* (510) — *Bord gauche* (520) — *Bord bas* (530) — *Raquette* (540) — *F augmente si on a tapé M et diminue si on a tapé Z* (800) — *Déplacement de la raquette* (840) — *Une autre partie* (920)

L'exemple qui suit montre une phase du jeu.



Pour écrire dans le fichier d'affichage il faut savoir que ce fichier comprend 24 lignes d'écriture, soit 2 lignes de plus que les lignes disponibles avec l'instruction PRINT, puisqu'un programme Basic réserve toujours 2 lignes dans le bas de l'écran.

Chaque ligne comprend 32 caractères, mais il existe un 33[e] caractère NEW LINE ; il faut faire très attention en écrivant directement dans le fichier d'affichage de ne pas écraser par une instruction POKE ce 33[e] caractère, car on bloquerait ainsi le fonctionnement du ZX 81, ce qui obligerait à couper l'alimentation pour remettre le micro-ordinateur en service.

2 — L'INSTRUCTION DE CHARGEMENT

Nous débuterons notre étude de la programmation en langage machine par l'utilisation de l'instruction de chargement LD. Cette instruction est celle qui est la plus utilisée dans la programmation en langage machine ; sous ses divers aspects cette instruction sert à transférer des informations d'un endroit à un autre.



TRANSFERT D'UNE DONNÉE DANS UNE CASE MÉMOIRE (1 K)

Notre premier programme machine comportera seulement trois instructions en langage machine. La première instruction LD charge un octet, c'est-à-dire un nombre compris entre 0 et 255, dans le registre accumulateur A du microprocesseur Z 80. La deuxième instruction LD transfère l'octet contenu dans le registre A à une adresse de la mémoire RAM que nous avons choisie.

La troisième instruction RET retourne au programme Basic.

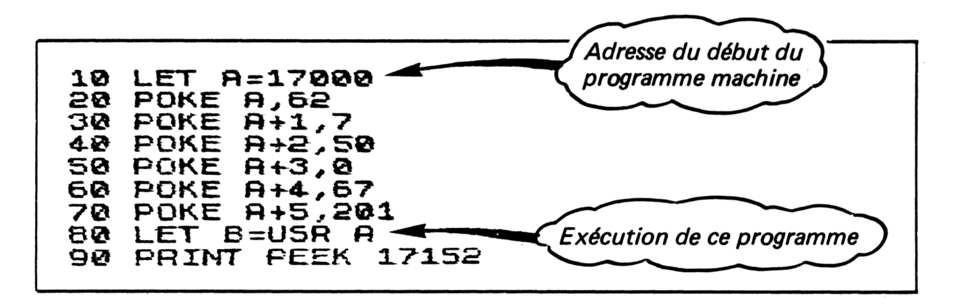
Nous voyons ci-dessous le programme écrit de trois manières différentes, en langage mnémonique-assembleur, en numération hexadécimale et en numération décimale.

Assembleur	**Hexadécimal**	**Décimal**
LD S,07	3E,07	62,07
LD 4300,A	32,00,43	50,0,67
RET	C 9	201

Ce programme machine va placer le nombre 7 dans la case de la mémoire RAM située à l'adresse 17 152 soit l'adresse 4300 en hexadécimal.

Pour entrer nos premiers programmes machine, dans la mémoire RAM du ZX 81 nous nous servirons d'une série d'instructions POKE, qui placeront les instructions en numération décimale dans les cases de la mémoire RAM ; par la suite nous verrons d'autres moyens d'entrer ces programmes machine dans la mémoire RAM.

Voici le programme Basic qui va entrer notre programme machine dans le ZX 81.




```
10 LET A=17000
20 POKE A,62
30 POKE A+1,7
40 POKE A+2,50
50 POKE A+3,0
60 POKE A+4,67
70 POKE A+5,201
80 LET B=USR A
90 PRINT PEEK 17152
```

La partie du programme qui crée le programme machine va de la ligne 20 à la ligne 70.

Lorsque ce programme est lancé nous voyons le nombre 7 s'afficher sur l'écran.

La ligne 10 du programme détermine l'adresse mémoire où commencera le programme machine.

La ligne 80 du programme provoque le passage du langage Basic en langage machine grâce à la fonction USR.

La ligne 90 affiche sur l'écran l'octet contenu à l'adresse 17 152, cet octet est le nombre 7 ; si à la ligne 30 nous remplaçons le 7 par un autre nombre c'est celui-ci qui s'affichera lorsque le programme sera lancé.



MODIFICATION D'UNE CASE MÉMOIRE (1 K)

Nous allons voir maintenant un nouveau programme qui comportera exactement les mêmes instructions machine que le précédent. Dans ce nouveau programme la donnée chargée à l'adresse 17152 est changée automatiquement à chaque cycle du programme.

```
  5 FOR C=1 TO 255
  8 CLS
 10 LET A=17000
 20 POKE A,62
 30 POKE A+1,C
 40 POKE A+2,50
 50 POKE A+3,0
 60 POKE A+4,67
 70 POKE A+5,201
 80 LET B=USR A
 90 PRINT PEEK 17152
100 PAUSE 40
110 NEXT C
```

Lorsque ce programme est lancé nous voyons les nombres de 1 à 255 s'afficher successivement sur l'écran.



CHARGEMENT DE DEUX OCTETS DANS LES REGISTRES B (1 K)

Il existe plusieurs manières de placer un programme en langage machine dans la mémoire vive du micro-ordinateur ZX 81, mais si on veut pouvoir enregistrer ce programme sur cassette, la méthode la plus pratique est de placer le programme dans une instruction REM au début du programme Basic.

Le programme machine débutera alors à l'adresse 16 514.

Cette adresse est celle du premier octet disponible dans un programme Basic.

Le programme machine qui va suivre sera placé de cette manière dans une instruction REM. Il charge un nombre de deux octets dans la paire de registres BC du microprocesseur, puis ce nombre est affiché sur l'écran.

Le nombre choisi est 3890 en décimal ou 0F32 en hexadécimal. Voici ce programme en assembleur, en numération hexadécimale et en numération décimale.

Assembleur	**Hexadécimal**	**Décimal**
LD B,0F	06,0F	6,15
LD C,32	0E,32	14,50
RET	C9	201

Ce programme machine comporte trois instructions ; la première charge l'octet 0F dans le registre B, la deuxième charge l'octet 32 dans le registre C et la troisième retourne au langage Basic.

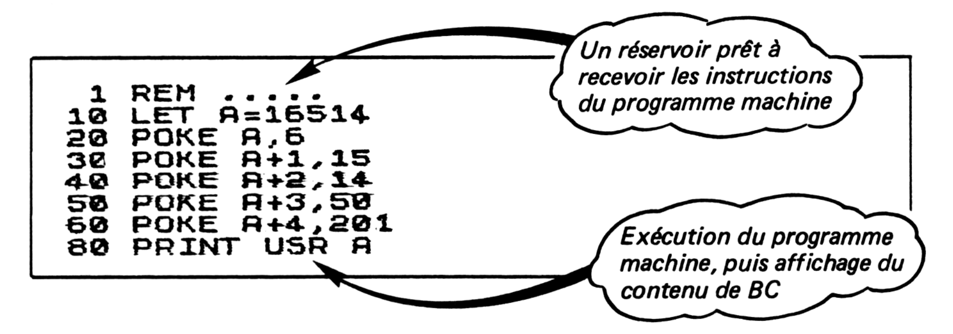
Voici le programme Basic chargeur qui place le programme machine dans une instruction REM.

Il débute à la ligne 1 pour une instruction REM suivie de 5 points ; le programme machine va venir prendre la place de ces points. Il doit y avoir au minimum autant de points que le programme machine comporte d'octets, mais le type de caractères placés à la suite de l'instruction REM n'a guère d'importance car ces caractères sont destinés à disparaître lorsqu'ils seront remplacés par le programme machine. A la place des points on aurait aussi bien pu utiliser les chiffres de 1 à 5.

La ligne 10 du programme fixe le début du programme machine à l'adresse 16514.

Les lignes 20 à 60 chargent le programme machine dans la mémoire RAM.

La ligne 80 provoque le passage du langage Basic au langage machine et affiche sur l'écran le contenu des registres B et C.



Lorsque nous lançons ce programme nous voyons s'afficher sur l'écran le nombre 3 890 que nous avons placé dans la paire de registres B,C à l'aide du programme machine.

Lorsque nous regardons à nouveau notre programme, ce qui est encore plus intéressant, est le fait que les points que nous avions placés après l'instruction REM ont disparu et sont remplacés par des caractères graphiques ou typographiques. Si vous recherchez la signification de ces caractères dans l'annexe A du manuel SINCLAIR vous verrez qu'ils correspondent aux octets que nous avons placés dans notre programme machine.

Le programme en langage machine est maintenant contenu dans l'instruction REM de la ligne 1 du programme Basic, ainsi qu'on peut le constater, si on demande le texte du programme après une première exécution :

```
 1 REM ▞?:MTAN
10 LET A=16514
20 POKE A,6
30 POKE A+1,15
40 POKE A+2,14
50 POKE A+3,50
60 POKE A+4,201
80 PRINT USR A
```

Non, ce ne sont pas des injures, mais le programme machine !

Comme l'instruction REM de la ligne 1 contient maintenant le langage machine, nous pouvons supprimer les lignes 20 à 60 du programme Basic chargeur devenues inutiles. Le programme réduit ci-après donne le même résultat que le programme initial.

```
 1 REM ▞?:MTAN
10 LET A=16514
80 PRINT USR A
```

3 — L'INSTRUCTION D'INCRÉMENTATION

Nous allons voir l'utilisation de l'instruction d'incrémentation INC, en langage machine.

Cette instruction permet d'incrémenter d'une unité l'octet contenu dans l'un des registres du microprocesseur Z 80. Cette instruction machine est semblable à l'instruction Basic

LET B = B + 1



INCRÉMENTATION D'UN OCTET (1 K)

Le programme qui suit va nous permettre de voir d'autres aspects de l'instruction machine LD, comme le transfert d'un registre à un autre.

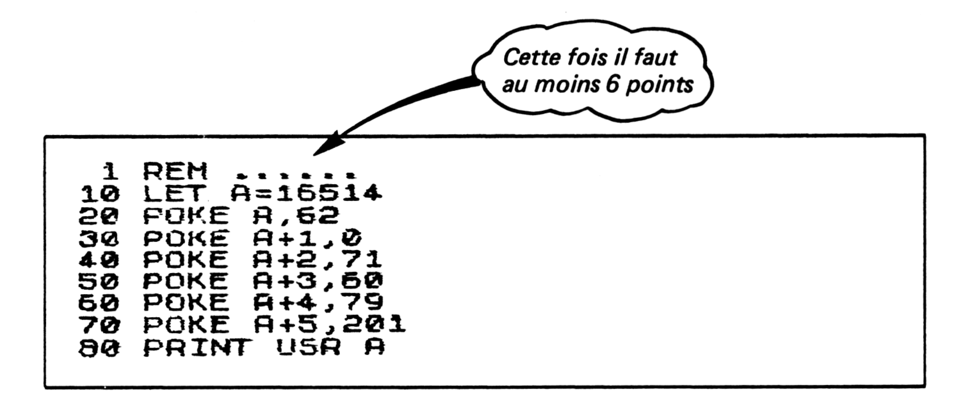
Ce programme charge l'octet zéro dans le registre A du microprocesseur Z 80 puis, cet octet est transféré dans le registre B, ensuite, l'octet zéro toujours contenu dans le registre A est incrémenté d'une unité avant d'être transféré dans le registre C. La paire de registres BC contient alors 1 c'est-à-dire 0 incrémenté d'une unité.

Voici ce programme qui comprend 5 instructions machine :

Assembleur	**Hexadécimal**	**Décimal**
LD A,00	3E,00	62,0
LD B,A	47	71
INC A	3C	60
LD C,A	4F	79
RET	C9	20 1

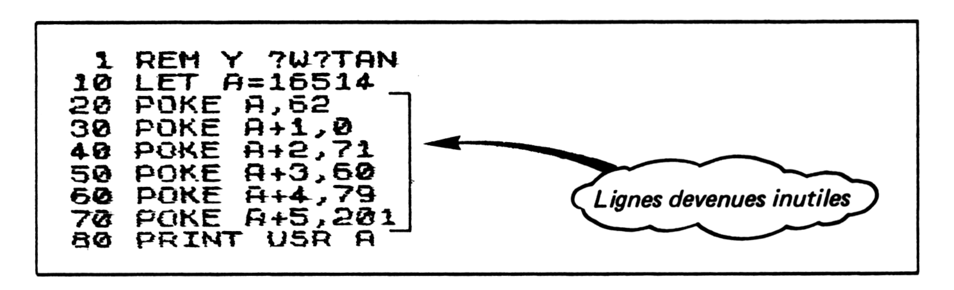
La première instruction charge l'octet 0 dans le registre A, la deuxième transfère le 0 dans le registre B, la troisième incrémente l'octet contenu dans A qui devient ainsi 1, la quatrième transfère cet octet dans le registre C, la cinquième retourne au langage Basic.

Voici le programme Basic chargeur.




```
 1 REM ......
10 LET A=16514
20 POKE A,62
30 POKE A+1,0
40 POKE A+2,71
50 POKE A+3,60
60 POKE A+4,79
70 POKE A+5,201
80 PRINT USR A
```

Lorsque ce programme est lancé nous voyons le chiffre 1, correspondant au zéro incrémenté, s'afficher sur l'écran. C'est le contenu de la paire de registres BC de la façon suivante :




```
 1 REM Y ?W?TAN
10 LET A=16514
20 POKE A,62
30 POKE A+1,0
40 POKE A+2,71
50 POKE A+3,60
60 POKE A+4,79
70 POKE A+5,201
80 PRINT USR A
```

Le programme machine est maintenant contenu dans l'instruction REM de la ligne 1 du programme Basic.

Les lignes 20 à 70 du programme sont devenues inutiles et peuvent être supprimées ; il ne reste alors qu'un programme réduit à 3 lignes donnant le même résultat que le programme primitif.

```
 1 REM Y ?W?TAN
10 LET A=16514
80 PRINT USR A
```

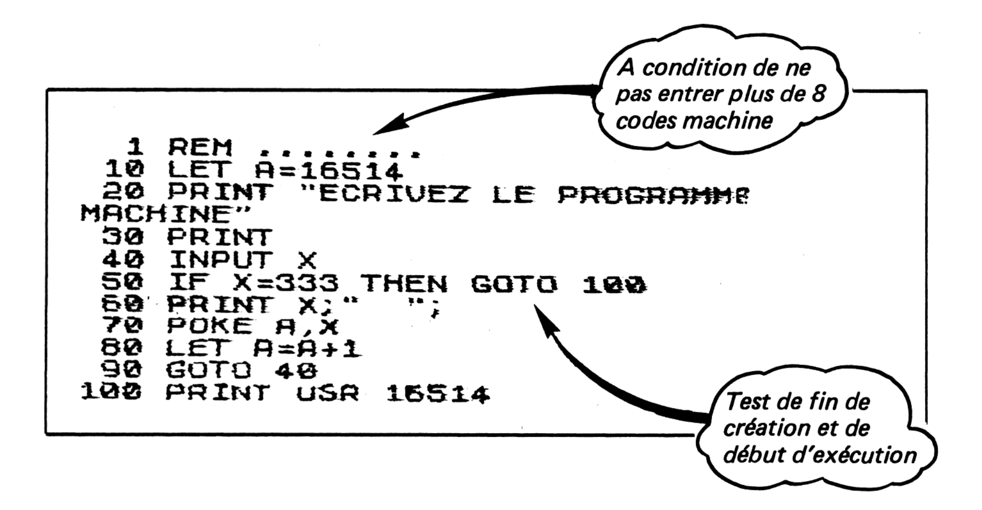
A l'instruction INC qui incrémente un registre correspond une instruction opposée DEC qui décrémente un registre. Le registre A est décrémenté par l'instruction DEC A qui a pour code opératoire 3D en hexadécimal et 61 en décimal.

A la ligne 50 du programme, remplaçons l'instruction d'incrémentation 60 par une instruction de décrémentation 61. Lorsque nous lançons le programme, nous voyons s'afficher 255 sur l'écran car un octet égal à zéro passe à 255 lorsqu'il est décrémenté d'une unité.



9 CHARGEUR DE CODES OPÉRATOIRES DÉCIMAUX (1 K)

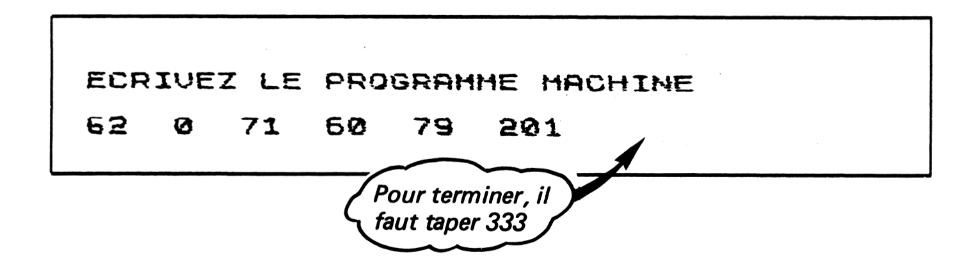
Lorsqu'un programme machine est un peu important, écrire un grand nombre de lignes avec des instructions POKE devient vite fastidieux. Le programme suivant permet d'entrer plus facilement les programmes machine dans la mémoire RAM.




```
  1 REM ........
 10 LET A=16514
 20 PRINT "ECRIVEZ LE PROGRAMME
MACHINE"
 30 PRINT
 40 INPUT X
 50 IF X=333 THEN GOTO 100
 60 PRINT X;"  ";
 70 POKE A,X
 80 LET A=A+1
 90 GOTO 40
100 PRINT USR 16514
```

Lorsque ce programme est lancé une inscription sur l'écran vous invite à écrire votre programme machine. Le programme s'affiche sur l'écran au fur et à mesure qu'il est écrit. Lorsqu'il est complété, il suffit d'écrire le nombre 333 pour provoquer l'exécution du programme machine.

L'exemple suivant montre l'affichage sur l'écran lorsqu'on vient d'entrer le programme machine précédent.




```
ECRIVEZ LE PROGRAMME MACHINE
62   0   71   60   79   201
```

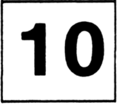
Comme pour les autres programmes Basic qui chargent des instructions en langage machine dans une instruction REM, une fois que ce programme est exécuté, c'est-à-dire après que l'on ait écrit le nombre 333, le programme machine se trouve placé dans l'instruction REM à la ligne 1. Il est possible de supprimer les lignes devenues inutiles comme le montre l'exemple suivant.

```
  1 REM Y ?W?TAN
 10 LET A=16514
 20 PRINT "ECRIVEZ LE PROGRAMME
MACHINE"
 30 PRINT
 40 INPUT X
 50 IF X=333 THEN GOTO 100
 60 PRINT X;"  ";
 70 POKE A,X
 80 LET A=A+1
 90 GOTO 40
100 PRINT USR 16514
```

On supprime ces lignes ...

... et voici le résultat

```
  1 REM Y ?W?TAN
 10 LET A=16514
100 PRINT USR 16514
```



DÉCODEUR DES CODES OPÉRATOIRES DÉCIMAUX (1 K)

Lorsqu'un programme en langage machine est contenu dans une instruction REM comme dans l'exemple précédent, ce programme se présente sous la forme d'une série de caractères graphiques difficiles à déchiffrer.

Le programme qui suit a pour but de décoder les programmes machine contenus dans des instructions REM pour reconstituer la liste des codes décimaux de ce programme.

Ce programme décodeur comprend trois lignes en langage Basic de la ligne 1000 à la ligne 1020. Il se place à la suite du programme machine que l'on désire reconstituer.

L'exemple qui suit montre ce programme de décodage écrit à la suite du programme machine précédent.

Il y a 6 octets à déchiffrer

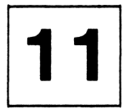
```
   1 REM Y ?W?TAN
  10 LET A=16514
 100 PRINT USR 16514
1000 FOR C=0 TO 5
1010 PRINT (16514+C);"..";PEEK
16514+C)
1020 NEXT C
```

Lorsque ce programme de décodage du langage machine est lancé par un GOTO 1000, l'on voit immédiatement se reconstituer le programme machine sur l'écran avec les adresses où sont placés les différents octets de ce programme.

```
16514..62
16515..0
16516..71
16517..60
16518..79
16519..201
```

La ligne 1000 doit être ajustée à l'importance du programme machine. Si ce programme avait comporté 32 octets, cette ligne aurait été FOR C = 0 TO 31.

4 — ADDITIONS



ADDITION DE DEUX NOMBRES DE UN OCTET (1 K)

L'instruction machine ADD est utilisée pour les additions. Nous allons voir un programme d'application de cette instruction qui va additionner les deux nombres 5 et 8.

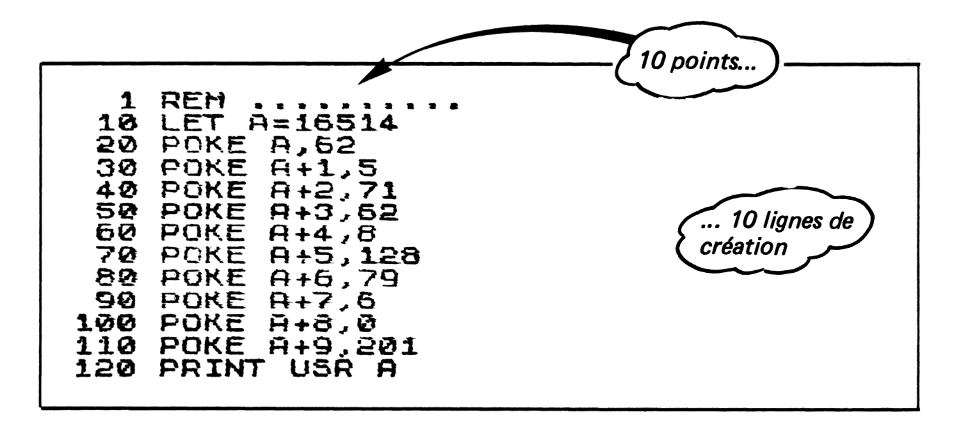
Assembleur	Hexadécimal	Décimal
LD A,05	3E,05	62,5
LD B,A	47	71
LD A,08	3E,08	62,8
ADD A,B	80	128
LD C,A	4F	79
LD B,00	06,00	6,0
RET	C9	201

Ce programme comporte 7 instructions machine, il est donc un peu plus important que les précédents. Vous remarquez, en plus de l'instruction d'addition ADD, les différentes formes de l'instruction de chargement LD.

La première instruction charge le nombre 5 dans le registre A du microprocesseur, la deuxième copie le contenu du registre A dans le registe B, la troisième charge la donnée 8 dans le registre A, et la quatrième additionne les octets contenus dans les registres A et B. Le résultat de l'opération se trouve dans le registre A. La cinquième instruction copie le contenu de A dans C, la sixième charge la valeur 0 dans le registre B et la septième retourne au langage Basic.

Lorsque ce programme machine est exécuté, le microprocesseur Z 80 effectue l'addition des deux nombres 5 et 8 et place le résultat de l'opération dans la paire de registres BC.

Voici le programme d'addition complet.




```
  1 REM ..........
 10 LET A=16514
 20 POKE A,62
 30 POKE A+1,5
 40 POKE A+2,71
 50 POKE A+3,62
 60 POKE A+4,8
 70 POKE A+5,128
 80 POKE A+6,79
 90 POKE A+7,6
100 POKE A+8,0
110 POKE A+9,201
120 PRINT USR A
```

Lorsque ce programme est lancé, la ligne 120 provoque l'exécution du programme machine et l'affichage sur l'écran du contenu de la paire de registres B,C. Ces registres contiennent le résultat de l'addition des nombres 5 et 8, aussi voit-on s'afficher le nombre 13 sur l'écran.

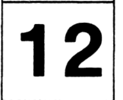
Lorsque l'on regarde à nouveau ce programme on voit que le programme machine a remplacé les points dans l'instruction REM du début du programme.

```
  1 REM Y▌?Y▒█?▟ TAN
 10 LET A=16514
 20 POKE A,62
 30 POKE A+1,5
 40 POKE A+2,71
 50 POKE A+3,62
 60 POKE A+4,8
 70 POKE A+5,128
 80 POKE A+6,79
 90 POKE A+7,6
100 POKE A+8,0
110 POKE A+9,201
120 PRINT USR A
```

Il devient alors possible de supprimer les lignes inutiles du programme, qui en étant réduit à trois lignes donne toujours le même résultat :

```
  1 REM Y▌?Y▒█?▟ TAN
 10 LET A=16514
120 PRINT USR A
```

Si aux lignes 30 et 60 du programme nous remplaçons 5 et 8 par deux autres nombres, c'est le résultat de l'addition de ces deux nombres qui s'affiche sur l'écran, à condition que leur somme ne dépasse pas 255.



ADDITION DE DEUX NOMBRES DE DEUX OCTETS (1 K)

Nous allons voir un nouveau programme machine d'addition de deux nombres, mais cette fois l'opération portera sur des nombres de deux octets.

Ce programme additionnera les nombres 17185 et 12834 c'est-à-dire les nombres 4321 et 3222 en hexadécimal.

Assembleur	**Hexadécimal**	**Décimal**
LD DE,4321	11,21,43	17,33,67
LD HL,3222	21,22,32	33,34,50
ADD HL,DE	19	25
LD B,H	44	68
LD C,L	4D	77
RET	C9	201

Dans ce programme, vous remarquerez les instructions machine de chargement et d'addition par paire de registres.

Le programme comporte 6 instructions machine, la première charge le nombre hexadécimal 4321 dans la paire de registres D,E ; ce chargement se fait en inversant les octets de ce nombre. La deuxième instruction charge le nombre hexadécimal 3222 dans la paire de registes HL, la troisième additionne le contenu des deux paires de registres HL et DE, la quatrième copie le contenu du registre H dans B, la cinquième copie le contenu du registre L dans C. La dernière instruction retourne au langage Basic.

Voici ce programme d'addition de deux nombres de deux octets, à condition que leur somme ne dépasse pas elle-même deux octets.

```
  1 REM ..........
 10 LET A=16514
 20 POKE A,17
 30 POKE A+1,33
 40 POKE A+2,67
 50 POKE A+3,33
 60 POKE A+4,34
 70 POKE A+5,50
 80 POKE A+6,25
 90 POKE A+7,68
100 POKE A+8,77
110 POKE A+9,201
120 PRINT USR A
```

Lorsque ce programme est lancé on voit s'afficher sur l'écran 30019 qui est bien le résultat de l'addition des deux nombres 17185 et 12834 que nous avions placés dans le programme en langage machine. Nous voyons aussi que le programme machine s'est logé dans l'instruction REM au début du programme en remplacement des points que nous y avions placés.

```
  1 REM )5?56M;??TAN
 10 LET A=16514
 20 POKE A,17
 30 POKE A+1,33
 40 POKE A+2,67
 50 POKE A+3,33
 60 POKE A+4,34
 70 POKE A+5,50
 80 POKE A+6,25
 90 POKE A+7,68
100 POKE A+8,77
110 POKE A+9,201
120 PRINT USR A
```

Si nous le désirons nous pouvons supprimer les lignes 20 à 110 de ce programme qui sont devenues inutiles, le programme donnera le même résultat, dans les mêmes conditions : la somme ne doit pas dépasser 65535 (le nombre maximum qui puisse tenir dans 2 octets).

5 — LES SAUTS DE PROGRAMME

En langage Basic les instructions GOTO et GOSUB permettent des sauts de programme et des bouclages de manière à exécuter des actions répétitives.

En langage machine il existe des fonctions semblables que le programme Basic ne nous permet pas d'apprécier.

13 AFFICHAGE D'ÉTOILES (1 K)

Ce programme en langage machine qui comporte seulement trois instructions va remplir d'étoiles la surface de l'écran.

Assembleur	Hexadécimal	Décimal
LD A,17	3E,17	62,23
CALL 08,08	CD,08,08	205,8,8
JR,F9	18,F9	24,249

C'est une boucle

La première instruction machine charge dans le registre accumulateur la donnée 17 en hexadécimal ou 23 en décimal, cette donnée représente le dessin de l'étoile. Vous pouvez remplacer cette étoile par un autre graphisme de votre choix en consultant l'annexe A du manuel SINCLAIR.

La deuxième instruction est une instruction de saut qui passe la main à un sous-programme placé dans la mémoire ROM à l'adresse 808 en hexadécimal ou 2056 en décimal. Ce sous-programme affiche sur l'écran le graphisme correspondant à l'octet contenu dans le registre A.

La troisième instruction est aussi une instruction de saut qui reboucle le programme à son début ; si cette instruction JR était remplacée par une instruction RET comme celle qui termine les programmes précédents, une seule étoile serait affichée sur l'écran.

Voici le programme chargeur :

```
  1 REM .......
 10 LET A=16514
 20 POKE A,62
 30 POKE A+1,23
 40 POKE A+2,205
 50 POKE A+3,8
 60 POKE A+4,8
 70 POKE A+5,24
 80 POKE A+6,249
 90 LET B=USR A
```

Il ne faut surtout pas utiliser PRINT ici : ça perturberait les étoiles !

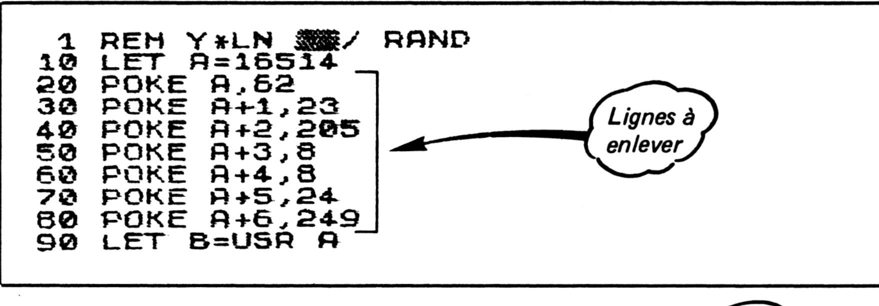
Lorsque ce programme est lancé, l'écran se remplit d'étoiles beaucoup plus rapidement qu'avec des instructions PRINT en langage Basic.

L'exemple suivant montre l'affichage des étoiles lorsque ce programme est utilisé sur un micro-ordinateur ZX 81 ayant 1 K de mémoire RAM.

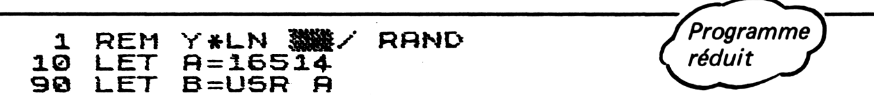
```
******************************
******************************
******************************
******************************
******************************
******************************
******************************
******************************
******************************
******************************
******************************
******************************
******************************
******************************
******************************
******************************
******************************
******************************
***********************
```

Cet affichage est incomplet à cause du manque de place dans la mémoire RAM. Si, pour ce même programme nous équipons notre ZX 81 d'un module d'extension 16 K mémoire, l'écran sera complètement recouvert d'étoiles.

Comme avec les programmes précédents, lorsque ce programme a été lancé une première fois, le langage machine se retrouve transposé dans l'instruction REM en tête du programme. Il devient alors possible de supprimer les lignes 20 à 90 du programme ce qui fait gagner de la place dans la mémoire RAM. On obtient ainsi le programme réduit à 3 lignes :



```
 1 REM Y*LN ▒/ RAND
10 LET A=16514
20 POKE A,62
30 POKE A+1,23
40 POKE A+2,205
50 POKE A+3,8
60 POKE A+4,8
70 POKE A+5,24
80 POKE A+6,249
90 LET B=USR A
```

```
 1 REM Y*LN ▒/ RAND
10 LET A=16514
90 LET B=USR A
```

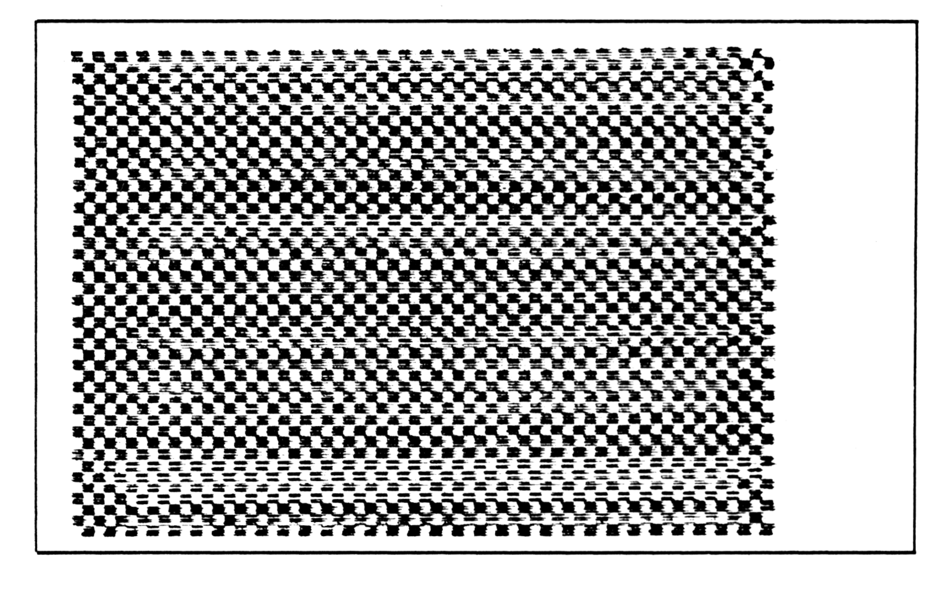

Lorsque ce programme réduit est lancé, l'écran se remplit complètement d'étoiles, même avec un ZX 81 ne comportant que 1 K de mémoire.

Lorsqu'un programme machine est placé dans une instruction REM il est toujours possible de modifier ce programme machine à l'aide de la fonction EDIT du ZX 81.

L'exemple suivant montre le programme réduit précédent dans lequel le deuxième caractère de l'instruction REM qui représentait l'étoile est remplacé par le caractère graphique (▚) de la touche Y.

```
 1 REM Y▚LN ▒/ RAND
10 LET A=16514
90 LET B=USR A
```

Lorsque ce programme réduit et modifié est lancé, l'affichage de l'écran est complètement différent.



14 AFFICHAGE DE DEUX CARACTERES (1 K)

Le programme qui suit est semblable au précédent, mais, au lieu d'afficher un seul caractère graphique sur l'écran, nous allons cette fois en afficher deux.

Voici ce programme en langage machine.

Assembleur	Hexadécimal	Décimal
LD A,B4	3E,B4	62,180
CALL 08,08	CD,08,08	205,8,8
LD B,80	06,80	6,128
ADD A,B	80	128
JR,F8	18,F8	24,248

Ce programme machine est analogue au précédent mais il comporte deux instructions de plus.

La première instruction charge la donnée B4 dans le registre A : cette donnée correspond à la lettre O en vidéo inversée.

La deuxième renvoie au sous-programme en mémoire ROM qui affiche sur l'écran le caractère qui correspond à l'octet que l'on vient de placer dans le registre A.

La troisième charge la donnée 80 dans le registre B.

La quatrième additionne le contenu des registres A et B. Après cette instruction, l'octet contenu dans le registre A correspond au même caractère mais en inversé.

La dernière instruction correspond à l'instruction GOTO du Basic : elle donne l'ordre de revenir en arrière de huit cases mémoire, le programme se rebouclant ainsi sur lui-même. Les instructions de saut relatif peuvent être aussi bien positives que négatives, ainsi 18,02 donnerait l'ordre de sauter 2 cases mémoires en avant et 18,FD donnerait l'ordre de sauter 3 cases mémoire en arrière.

Le programme complet donné ci-après, remplit l'écran alternativement de lettres O en vidéo normale et inversée.

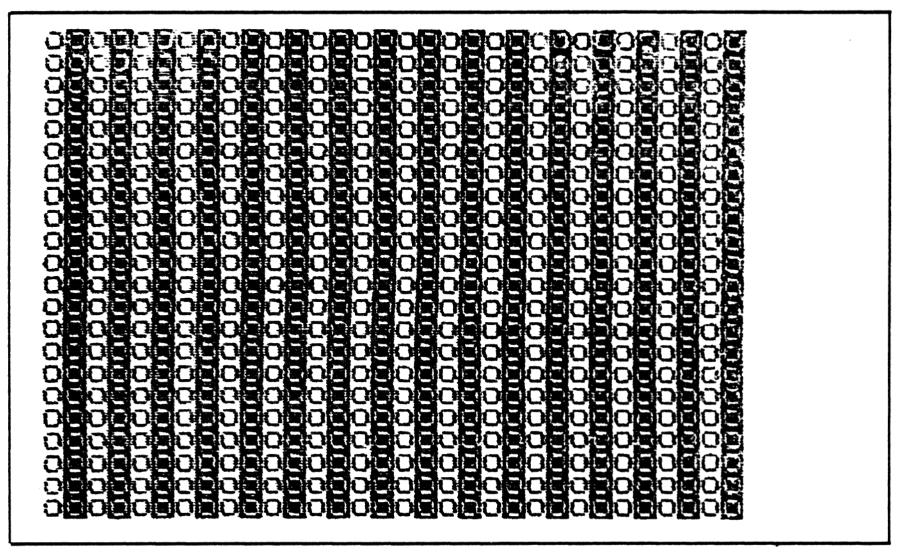
```
   1 REM ..........
  10 LET A=16514
  20 POKE A,62
  30 POKE A+1,180
  40 POKE A+2,205
  50 POKE A+3,8
  55 POKE A+4,8
  60 POKE A+5,6
  70 POKE A+6,128
  80 POKE A+7,128
  90 POKE A+8,24
 100 POKE A+9,248
 110 LET B=USR A
```

Lorsque ce programme est lancé pour la première fois le programme machine vient se placer dans l'instruction REM du début du programme, ce qui permet d'effacer les lignes 20 à 100 du programme devenues désormais inutiles et ainsi de gagner de la place dans la mémoire RAM.

L'exemple suivant montre ce programme ainsi réduit à trois lignes.

```
   1 REM Y█LN ▒▒▞██/ SAVE
  10 LET A=16514
 110 LET B=USR A
```

Lorsque ce programme réduit est lancé on voit s'afficher sur l'écran des colonnes de lettres O en vidéo inversée et des colonnes en vidéo normale.



A l'aide de la fonction EDIT du ZX 81 il est possible de remplacer le deuxième caractère de l'instruction REM du programme réduit. Ce caractère est la lettre O en vidéo inversée. Lorsque ce programme réduit sera lancé à nouveau c'est le nouveau caractère et son inverse qui viendront remplir l'écran.

6 — CHARGEUR HEXADECIMAL

15 PROGRAMME CHARGEUR HEXADÉCIMAL (1 K)

En langage machine, l'utilisation d'une série d'instructions POKE, pour entrer dans les cases de la mémoire RAM les codes opératoires de ces instructions en numération décimale, n'est valable que pour des programmes très courts.

Précédemment nous avons vu un programme en langage Basic qui chargeait dans la mémoire RAM des programmes machines exprimés en numération décimale mais, dès qu'un programme machine comporte plus de quelques octets, il est préférable de charger directement ce programme en numération hexadécimale.

Les instructions du microprocesseur Z 80 sont en effet données en numération hexadécimale. Pour charger ces instructions dans la mémoire RAM en numération décimale, il faut se livrer à une série de calculs fastidieux. Ceux-ci seront évités en entrant les programmes directement en numération hexadécimale.


```
  1 REM ......
 10 LET A=16514
 20 LET A$=""
 30 FOR B=1 TO LEN A$-1 STEP 2
 40 LET C=CODE A$(B)-28
 50 LET D=CODE A$(B+1)-28
 60 POKE A,16*C+D
 70 LET A=A+1
 80 NEXT B
100 PRINT USR 16514
```

Ce programme doit être sauvé sur cassette car il nous servira constamment pour entrer les programmes en langage machine. Nous ne traduisons plus les instructions en numération décimale. Ce programme de chargement débute par une instruction REM, qui contiendra le programme machine après un premier lancement.

Cette instruction REM est suivie d'un nombre de points, qui doit être au moins égal au nombre d'octets que comporte le programme machine.

La ligne 20 du programme de chargement comporte une chaîne de caractères, qui au départ est vide. C'est dans cette chaîne de caractères que nous implanterons notre programme machine.

16 AFFICHAGE DE 9 CARACTERES (1 K)

Nous allons voir un programme machine qui affiche sur l'écran les neuf premiers caractères typographiques du micro-ordinateur ZX 81.

Assembleur	**Hexadécimal**
LD A,09	3E,09
CALL 08,08	CD,08,08
DEC,A	3D
JRNZ,FA	20,FA
RET	C9

Une boucle

Ce programme comporte 5 instructions machines. La première instruction charge la donnée 09 dans le registre accumulateur A, cette donnée correspond au nombre de caractères qui sera affiché sur l'écran. Si nous remplaçons cette donnée par un autre nombre, c'est ce nombre de caractères qui sera alors affiché sur l'écran.

La deuxième affiche sur l'écran le graphisme correspondant à l'octet contenu dans le registre A.

La troisième instruction décrémente d'une unité le contenu du registre A (si ce contenu est 9 il devient 8).

La quatrième instruction est une instruction de saut conditionnel qui commande au programme de revenir en arrière de 6 cases mémoire au niveau de la deuxième instruction si le contenu du registre accumulateur est différent de zéro.

La dernière instruction retourne au langage Basic.

Voici le programme chargeur avec le programme machine placé dans la chaîne de caractères de la ligne 20.

Lorsque ce programme est lancé les neuf premiers caractères du ZX 81 viennent s'afficher sur l'écran.

Le programme machine

```
  1 REM .........
 10 LET A=16514
 20 LET A$="3E09CD08083D20FAC9"
 30 FOR B=1 TO LEN A$-1 STEP 2
 40 LET C=CODE A$(B)-28
 50 LET D=CODE A$(B+1)-28
 60 POKE A,16*C+D
 70 LET A=A+1
 80 NEXT B
100 PRINT USR 16514
```

L'exemple suivant montre les 9 caractères graphiques qui s'affichent sur l'écran.

```
▒▒▛▚▜ ▪ ▀ ▀ 0
```

Le zéro qui suit ces neuf caractères est le contenu de la paire de registres BC que le PRINT de la ligne 100 affiche sur l'écran. Si, par exemple, l'on remplace dans cette ligne PRINT par RAND ce zéro disparaît lorsque le programme est lancé. L'essentiel est de faire apparaître USR 16514 pour exécuter le programme machine.

Après le premier lancement, le programme en langage machine est implanté dans l'instruction REM, comme pour les autres programmes ; il est alors possible de supprimer les lignes du programme devenues inutiles, c'est-à-dire les lignes 10 à 80. Il reste alors un programme de deux lignes donnant le même résultat.

```
  1 REM Y▒LN ▒X4 IF TAN
100 PRINT USR 16514
```

Si nous remplaçons le deuxième caractère de l'instruction REM représentant le nombre 9 par le chiffre 0 qui correspond au nombre 28, lorsque ce programme réduit sera lancé ce sont les 28 premiers caractères qui seront affichés.

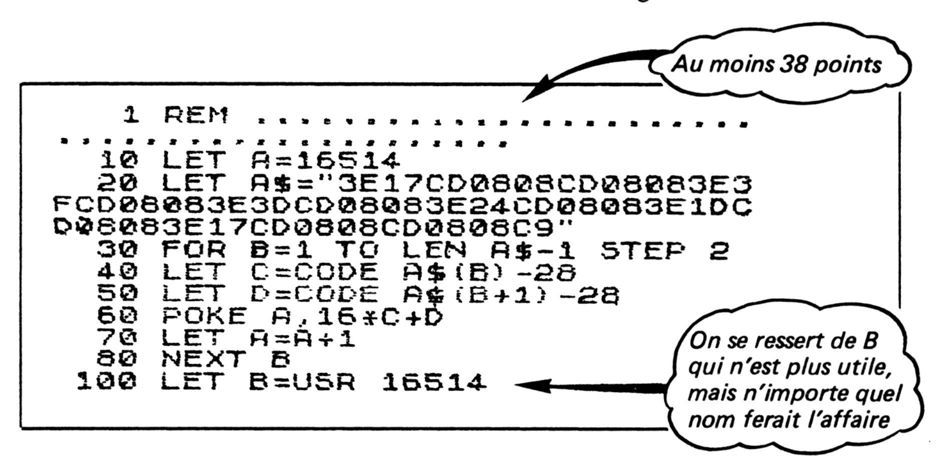
17 AFFICHAGE DE ** ZX 81 **

Le programme machine qui suit va afficher sur l'écran ZX 81 entre quatre astérisques. Il serait possible de rendre ce programme plus compact, mais alors le programme deviendrait moins compréhensible.

Ce programme machine comporte 38 octets et l'instruction REM à la ligne 1 du programme chargeur doit comporter au moins autant de points.

Hexadécimal	Assembleur	
3E,17	LD A,17	*17 correspond à l'astérisque*
CD,08,08	CALL 08,08	*imprime le 1er astérisque*
CD,08,08	CALL 08,08	*imprime le 2e astérisque*
3E,3F	LD A,3F	*3F correspond à Z*
CD,08,08	CALL 08,08	*imprime Z sur l'écran*
3E,3D	LD A,3D	*3D correspond à X*
CD,08,08	CALL 08,08	*imprime X*
3E,24	LD A,24	*24 correspond à 8*
CD,08,08	CALL 08,08	*imprime 8*
3E,1D	LD A,1D	*1D correspond à 1*
CD,08,08	CALL 08,08	*imprime 1*
3E,17	LD A,17	*17 correspond à l'astérisque*
CD,08,08	CALL 08,08	*imprime le 3e astérisque*
CD,08,08	CALL 08,08	*imprime le 4e astérisque*
C9	RET	*retourne au Basic*

L'exemple suivant montre le programme chargeur avec le programme machine contenu dans la chaîne de caractères de la ligne 20.




```
   1 REM ........................
......................
  10 LET A=16514
  20 LET A$="3E17CD0808CD08083E3
FCD08083E3DCD08083E24CD08083E1DC
D08083E17CD0808CD0808C9"
  30 FOR B=1 TO LEN A$-1 STEP 2
  40 LET C=CODE A$(B)-28
  50 LET D=CODE A$(B+1)-28
  60 POKE A,16*C+D
  70 LET A=A+1
  80 NEXT B
 100 LET B=USR 16514
```

Lorsque ce programme est lancé, le nom du ZX 81 s'inscrit sur l'écran entre 4 astérisques :

```
**ZX81**
```

Le programme machine s'est alors implanté dans l'instruction REM de la ligne 1 :

```
   1 REM Y*LN ▒LN ▒YZLN ▒YXLN
 ▒Y8LN ▒Y1LN ▒Y*LN ▒LN ▒TAN
 .......
  10 LET A=16514
  20 LET A$="3E17CD0808CD08083E3
FCD08083E3DCD08083E24CD08083E1DC
D08083E17CD0808CD0808C9"
  30 FOR B=1 TO LEN A$-1 STEP 2
  40 LET C=CODE A$(B)-28
  50 LET D=CODE A$(B+1)-28
  60 POKE A,16*C+D
  70 LET A=A+1
  80 NEXT B
 100 LET B=USR 16514
```

Programme réduit :

```
   1 REM Y*LN ▒LN ▒YZLN ▒YXLN
 ▒Y8LN ▒Y1LN ▒Y*LN ▒LN ▒TAN
 .......
 100 LET B=USR 16514
```

Dans ce programme réduit nous pouvons voir les graphismes des astérisques ; à l'aide de la fonction EDIT du ZX 81 nous pouvons modifier le programme machine en remplacant les astérisques par le signe + en vidéo inversée ainsi que le montre l'exemple suivant.

```
   1 REM Y+LN ▒LN ▒YZLN ▒YXLN
 ▒Y8LN ▒Y1LN ▒Y+LN ▒LN ▒TAN
 .......
 100 LET B=USR 16514
```

Voici le nouvel affichage obtenu sur l'écran :

```
++ZX81++
```

7 — LA PAIRE DE REGISTRES BC

18 AFFICHAGE DU NOMBRE 15 (1 K)

Lorsque l'instruction PRINT USR adresse est exécutée, le programme machine commençant à l'adresse indiquée est mis en service et le contenu de la paire de registres BC est affiché sur l'écran. La connaissance de ces registres est donc très utile.

Nous allons voir quelques programmes pour nous familiariser davantage avec ces registres B et C.

Le premier programme machine de trois instructions affiche simplement le nombre 15 sur l'écran.

Hexadécimal	**Assembleur**	
06,00	LD B,00	Charge 00 dans B
0E,0F	LD C,0F	Charge 0F (15 en décimal) dans C
C9	RET	Retourne au Basic

On voit ci-dessous le programme chargeur avec le programme machine contenu dans la chaîne de caractères de la ligne 20.

```
  1 REM .....
 10 LET A=16514
 20 LET A$="06000E0FC9"
 30 FOR B=1 TO LEN A$-1 STEP 2
 40 LET C=CODE A$(B)-28
 50 LET D=CODE A$(B+1)-28
 60 POKE A,16*C+D
 70 LET A=A+1
 80 NEXT B
100 PRINT USR 16514
```

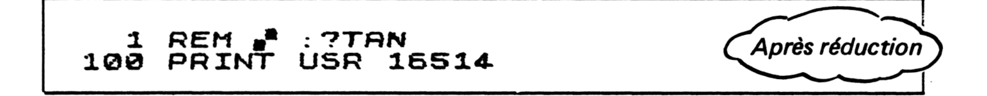
Avant la première exécution

Lorsque ce programme est lancé, le nombre 15 s'affiche sur l'écran et le programme machine vient se placer dans l'instruction REM de la ligne 1 comme le montre l'exemple suivant :

```
  1 REM ▞ :?TAN
 10 LET A=16514
 20 LET A$="06000E0FC9"
 30 FOR B=1 TO LEN A$-1 STEP 2
 40 LET C=CODE A$(B)-28
 50 LET D=CODE A$(B+1)-28
 60 POKE A,16*C+D
 70 LET A=A+1
 80 NEXT B
100 PRINT USR 16514
```

Après la première exécution

Il est alors possible de faire disparaître de ce programme les lignes 10 à 80 devenues inutiles. Il reste alors le programme réduit qui affiche toujours le nombre 15 :



```
  1 REM ▞ :?TAN
100 PRINT USR 16514
```


Les signes graphiques contenus dans l'instruction REM constituent le langage machine ; ces signes correspondent au code hexadécimal du langage machine que nous avons placé dans la chaîne de caractères du programme chargeur, comme nous pouvons nous en assurer en regardant l'annexe A du manuel SINCLAIR sur le micro-ordinateur ZX 81.

Il est possible de modifier ce programme machine en nous servant de la fonction EDIT du ZX 81. Le quatrième signe de ce programme (?) correspond au nombre 15 qui est affiché. Si nous remplaçons ce point d'interrogation par le signe + représentant le nombre 21, c'est alors 21 qui sera affiché lorsque le programme est lancé :

```
  1 REM ▞ :+TAN
100 PRINT USR 16514
```

19 DOUBLE INCRÉMENTATION (1 K)

Le programme machine qui suit comporte 5 instructions. Le nombre 20 en hexadécimal, ou 32 en décimal, va être chargé dans la paire de registres BC, puis C sera incrémenté deux fois de suite si bien que c'est le nombre 34 qui apparaîtra sur l'écran.

Voici le programme chargeur avec le programme machine contenu dans la chaîne de caractères de la ligne 20 :

```
  1 REM .......
 10 LET A=16514
 20 LET A$="06000E200C0CC9"
 30 FOR B=1 TO LEN A$-1 STEP 2
 40 LET C=CODE A$(B)-28
 50 LET D=CODE A$(B+1)-28
 60 POKE A,16*C+D
 70 LET A=A+1
 80 NEXT B
100 PRINT USR 16514
```

Après un premier lancement le programme machine est contenu dans l'instruction REM de la ligne 1 ; ceci permet de supprimer les lignes 10 à 80 du programme de chargement :

```
  1 REM ▞ :4££TAN
100 PRINT USR 16514
```

Si, dans ce programme réduit, nous supprimons les deux instructions machine d'incrémentation, qui sont représentées par le signe de la livre (£), à l'aide de la fonction EDIT, nous obtenons le nouveau programme réduit suivant qui affiche le nombre 32 sur l'écran.

```
  1 REM ▞ :4TAN
100 PRINT USR 16514
```

20 SOUSTRACTION (1 K)

Nous allons voir une soustraction en langage machine : 15-8 avec affichage du résultat sur l'écran.

Hexadécimal	Assembleur	
3E,0F	LD A,0F	*Charge 0F dans A soit 15 en décimal*
01,08,00	LD BC,0008	*Charge 8 dans BC*
99	SBC A,C	*Soustrait C de A*
4F	LD C,A	*Charge A dans C*
C9	RET	*Retourne au Basic*

L'exemple suivant montre le programme chargeur avec le programme machine contenu dans la chaîne de caractères de la ligne 20 :

```
  1 REM ........
 10 LET A=16514
 20 LET A$="3E0F010800994FC9"
 30 FOR B=1 TO LEN A$-1 STEP 2
 40 LET C=CODE A$(B)-28
 50 LET D=CODE A$(B+1)-28
 60 POKE A,16*C+D
 70 LET A=A+1
 80 NEXT B
100 PRINT USR 16514
```

Le résultat de l'opération, 7, s'affiche sur l'écran et le programme machine vient se placer dans l'instruction REM de la ligne 1, ce qui permet de supprimer les lignes 10 à 80.

```
  1 REM Y?▀▒ █?TAN
100 PRINT USR 16514
```

21 DOUBLE PROGRAMME MACHINE (1 K)

Il est possible dans un programme écrit en langage machine ou en Basic, d'avoir plusieurs sous-programmes en langage machine et de passer à l'un ou l'autre de ces sous-programmes lorsque cela est nécessaire.

Le programme machine qui suit comporte deux parties pouvant être utilisées séparément.

Hexadécimal	**Assembleur**	
06,00	LD B,00	*Charge 00 dans B*
0E,20	LD C,20	*Charge 20 soit 32 en décimal dans C*
C9	RET	*Retourne au Basic*
06,00	LD B,00	*Charge 00 dans B*
0E,40	LD C,40	*Charge 40 soit 64 en décimal dans C*
C9	RET	*Retourne au Basic*

L'exemple suivant montre le programme chargeur avec le programme machine placé dans la chaîne de caractères, de la ligne 20 :

```
   1 REM ...........
  10 LET A=16514
  20 LET A$="06000E20C906000E40C
9"
  30 FOR B=1 TO LEN A$-1 STEP 2
  40 LET C=CODE A$(B)-28
  50 LET D=CODE A$(B+1)-28
  60 POKE A,16*C+D
  70 LET A=A+1
  80 NEXT B
 100 PRINT USR 16514
 110 PRINT USR 16519
```

Ici on utilise les deux programmes machine l'un après l'autre

Lorsque ce programme est lancé, les deux parties du programme machine sont mises successivement en service par les deux instructions PRINT. Il en résulte un double affichage sur l'écran, 32 pour la première partie du programme machine à l'adresse 16 514 et 64 pour la deuxième partie à l'adresse 16 519, comme le montre une reproduction de cet affichage.

```
32
64
```

Après le premier lancement, le langage machine est contenu dans l'instruction REM de la ligne 1 et il devient possible de supprimer les lignes 10 à 80 du programme pour ne conserver que le programme réduit :

```
  1 REM ▞ :4TAN ▞ :RNDTAN
100 PRINT USR 16514
110 PRINT USR 16519
```

8 — UNE AUTRE INSTRUCTION D'AFFICHAGE

22 AFFICHAGE RÉPÉTITIF D'UN CARACTERE (1 K)

Nous avons vu que l'instruction en langage machine, CALL 0808, permettait d'afficher sur l'écran le caractère correspondant à un octet. Nous pouvons obtenir la même fonction à l'aide de l'instruction RST, soit en numération hexadécimale : D7.

Les programmes qui suivent vont utiliser cette instruction.

Le premier programme affiche d'une manière répétitive un caractère graphique afin de remplir l'écran.

Hexadécimal	**Assembleur**	
→ 3E,07	LD A,07	*Chargement du caractère*
D7	RST	*Affichage*
— 18,FD	JR,FD	*Saut à −3*

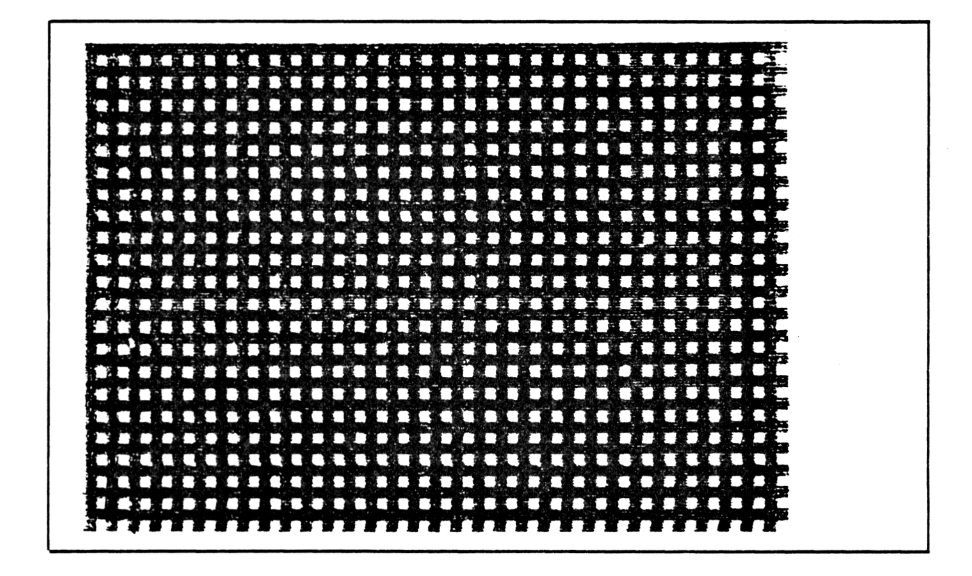
Le programme Basic :

```
  1 REM .....
 10 LET A=16514
 20 LET A$="3E07D718FD"
 30 FOR B=1 TO LEN A$-1 STEP 2
 40 LET C=CODE A$(B)-28
 50 LET D=CODE A$(B+1)-28
 60 POKE A,16*C+D
 70 LET A=A+1
 80 NEXT B
100 PRINT USR 16514
```

Programme en langage machine :

```
  1 REM Y▛NOT / CLEAR
100 PRINT USR 16514
```

Lorsque ce programme machine est lancé, on voit le caractère choisi s'affiche sur l'écran d'une manière répétitive pour former le dessin que montre l'exemple suivant :



A l'aide de la fonction EDIT il est facile de modifier le deuxième caractère du programme en langage machine, qui représente le caractère affiché sur l'écran pour le remplacer par un autre et obtenir ainsi un dessin différent.



AFFICHAGE DE PLUSIEURS CARACTERES (1 K)

Le programme en langage machine qui suit affiche sur l'écran un dessin qui rappelle le tissage des couvertures mexicaines.

Hexadécimal	Assembleur	
→3E,39	LD A,39	*Chargement du 1er caractère*
D7	RST	*Affichage*
3E,14	LD A,14	*Chargement du 2e caractère*
D7	RST	*Affichage*
3E,39	LD A,39	*Chargement du 3e caractère*
D7	RST	*Affichage*
3E,94	LD A,94	*Chargement du 4e caractère*
D7	RST	*Affichage*
└18,F2	JR,F2	*Saut à – 14*

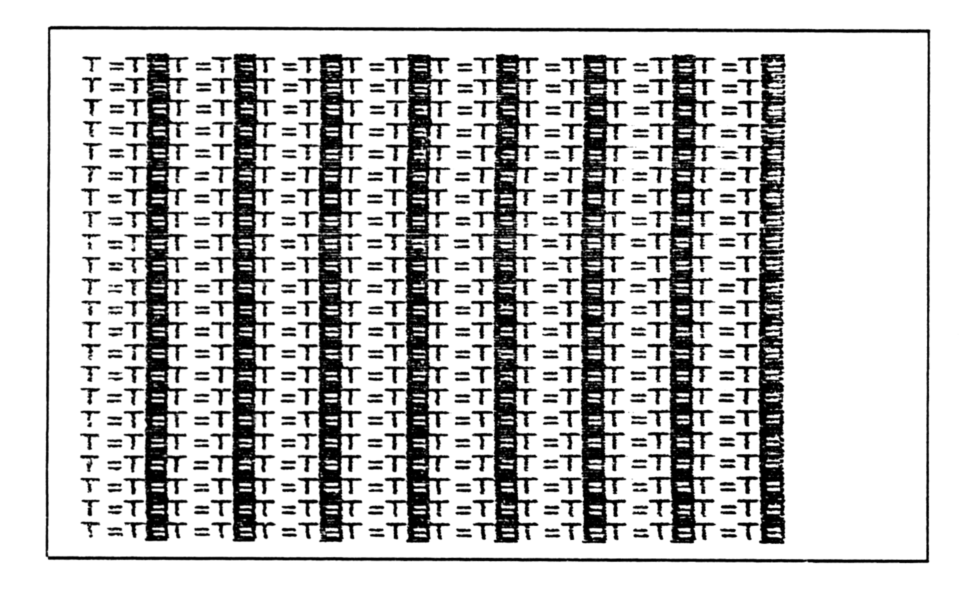
Programme Basic chargeur :

```
   1 REM ...............
  10 LET A=16514
  20 LET A$="3E39D73E14D73E39D73
E94D718F2"
  30 FOR B=1 TO LEN A$-1 STEP 2
  40 LET C=CODE A$(B)-28
  50 LET D=CODE A$(B+1)-28
  60 POKE A,16*C+D
  70 LET A=A+1
  80 NEXT B
 100 PRINT USR 16514
```

Programme en langage machine :

```
  1 REM YTNOT Y=NOT YTNOT Y█NOT
/ PAUSE .
100 PRINT USR 16514
```

Lorsque ce programme machine est lancé, voici le résultat :





EMPLOI D'UN COMPTEUR DE CARACTERES (1 K)

Dans les programmes que nous venons de voir, l'affichage du ou des mêmes caractères sur l'écran se répétait jusqu'à ce que l'écran ou la mémoire RAM soient pleins.

Le programme machine qui suit va afficher sur l'écran un certain nombre de fois un caractère, ensuite il fera de même pour un autre caractère.

Le nombre de fois que le caractère sera affiché sur l'écran sera déterminé par la valeur de l'octet qui sera placé dans un compteur.

Hexadécimal	**Assembleur**	
1E,80	LD E,80	*Chargement du 1er compteur*
3E,17	LD A,17	*Chargement du 1er caractère*
D7	RST	*Affichage*
1D	DEC,E	*Décrémentation du 1er compteur*
20,FC	JRNZ,FC	*Saut à –4 si le compteur n'est pas 0*
1E,80	LD E,80	*Chargement du 2e compteur*
3E,07	LD A,07	*Chargement du 2e caractère*
D7	RST	*Affichage*
1D	DEC,E	*Décrémentation du 2e compteur*
20,FC	JNRZ,FC	*Saut à –4 si le compteur n'est pas 0*
18,EE	JR,EE	*Saut à –18 si le compteur est nul*

Voici le programme de chargement avec le programme machine contenu dans la chaîne de caractères de ligne 20 :

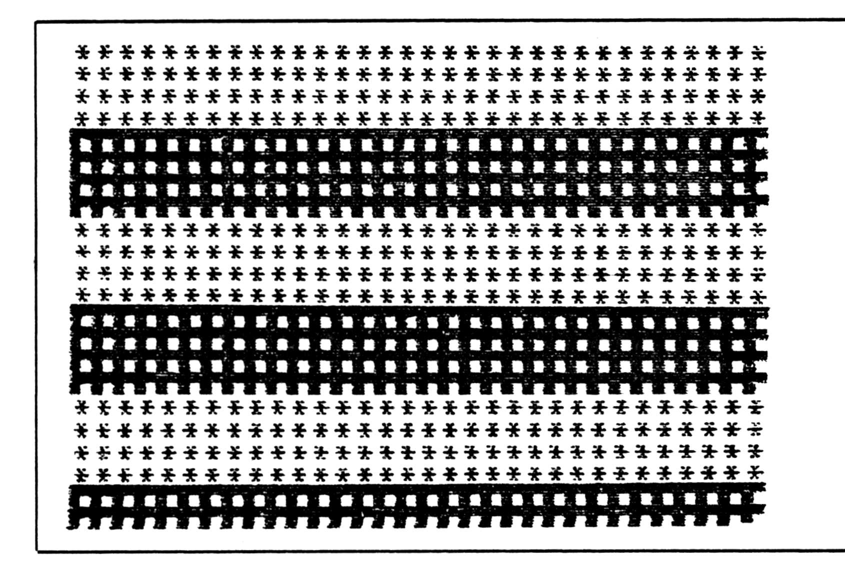
```
   1 REM ......................
  10 LET A=16514
  20 LET A$="1E803E17D71D20FC1E8
03E07D71D20FC18EE"
  30 FOR B=1 TO LEN A$-1 STEP 2
  40 LET C=CODE A$(B)-28
  50 LET D=CODE A$(B+1)-28
  60 POKE A,16*C+D
  70 LET A=A+1
  80 NEXT B
 100 PRINT USR 16514
```

Dans le programme machine il est possible de modifier les caractères affichés aussi bien que le contenu des compteurs.

Programme en langage machine :

```
    1 REM 2▇Y*NOT 14 UNPLOT 2▇Y▛N
OT 14 UNPLOT / INPUT ..
  100 PRINT USR 16514
```

Voici le résultat après lancement :



9. LES MOUVEMENTS RAPIDES SUR L'ECRAN

25 ÉTOILE FILANTE EN LANGAGE BASIC (1 K)

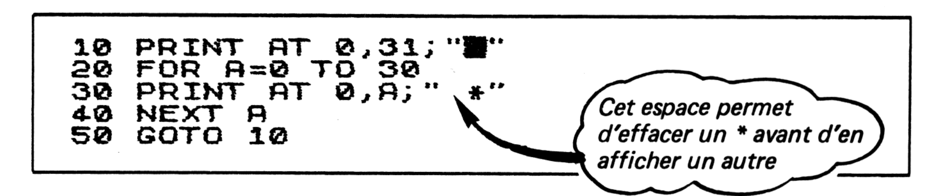
Nous savons que le langage machine permet une exécution des programmes bien plus rapides que lorsque le même programme est écrit en langage Basic.

Ceci est particulièrement important lorsque l'on programme des jeux qui exigent des mouvements sur l'écran. Bien souvent en langage Basic ces mouvements sont trop lents, mais l'utilisation d'un sous-programme en langage machine permet d'accélérer jusqu'à cinquante fois la rapidité de ces mouvements.

Pour illustrer ceci nous allons voir un programme très simple, que nous allons écrire de deux façons : en langage machine et en Basic.

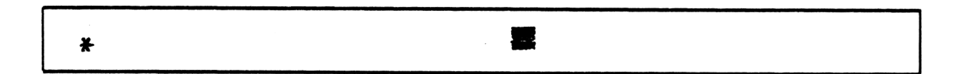
Ce programme affiche un carré noir en haut et à droite de l'écran, ensuite une étoile part de la gauche de l'écran et se dirige vers le carré noir ; une fois arrivée au carré noir elle saute instantanément à son point de départ, puis ce cycle est répété constamment. C'est la vitesse des mouvements de l'étoile que nous allons comparer en langage machine et en Basic.

Voici le programme Basic :



```
10 PRINT AT 0,31;"■"
20 FOR A=0 TO 30
30 PRINT AT 0,A;" *"
40 NEXT A
50 GOTO 10
```


Lorsque ce programme est lancé, on voit l'étoile se diriger vers le carré.



Il est possible de ralentir le mouvement de l'étoile en introduisant une boucle de temporisation dans le programme.

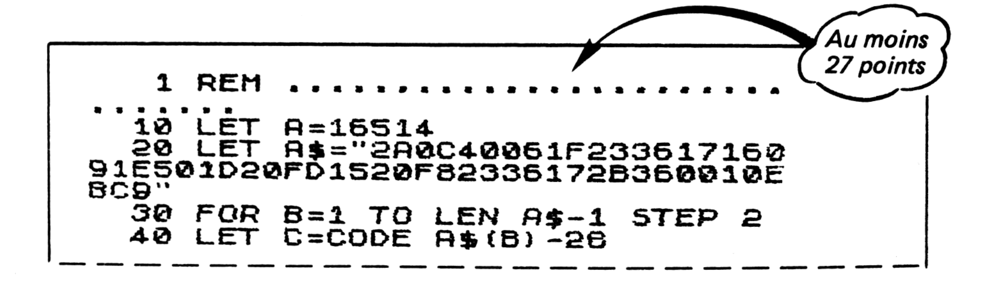
26 ETOILE FILANTE EN LANGAGE MACHINE (1 K)

Nous allons voir le programme machine qui donnera le même mouvement de l'étoile, mais dans ce programme il sera nécessaire d'introduire des boucles de temporisation destinées à ralentir le mouvement de l'étoile, qui, sans cela, seraient bien trop rapides pour être visibles.

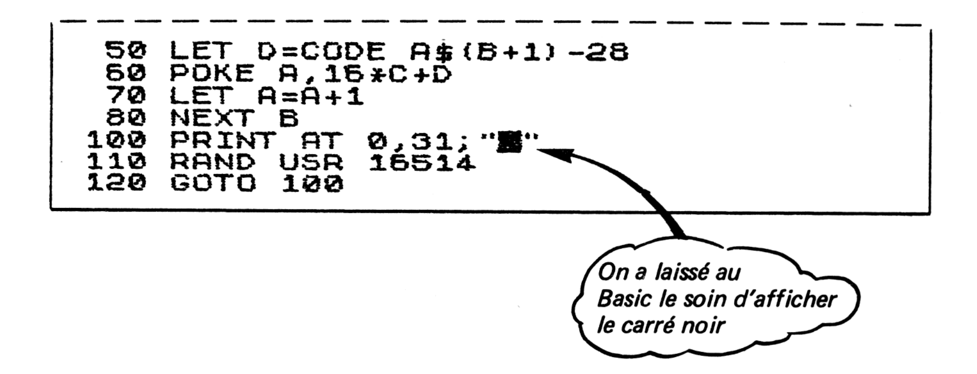
Hexadécimal	Assembleur	
2A,0C,40	LD HL, 40 0C	*Charge dans HL l'adresse d'affichage*
06,1F	LD B,IF	*Charge dans B le nombre de pas*
23	INC HL	*Avance d'un pas*
36,17	LD HL,17	*Affiche l'étoile*
16,09	LD D,09	*Charge temporisation 1*
1E,50	LD E,50	*Charge temporisation 2*
1D	DEC,E	*Décrémentation temporisation 2*
20,FD	JRNZ,FD	*Saut à −3 si non zéro*
15	DEC D	*Décrémentation temporisation 1*
20,F8	JRNZ,F8	*Saut à −8 si non zéro*
23	INC HL	*Avance d'un pas*
36,17	LD HL,17	*Affiche l'étoile*
2B	DEC HL	*Recule d'un pas*
36,00	LD HL,00	*Efface l'étoile précédente*
10,EB	DJNZ,EB	*Décrémente B et saute à −21 si non zéro*
C9	RET	*Retourne au Basic*

L'exemple suivant montre notre programme chargeur avec les trois dernières lignes légèrement modifiées, le programme machine étant placé dans la chaîne de caractères de la ligne 20.

Remarquez que la ligne 100 de ce programme et la ligne 10 du programme Basic sont identiques.




```
   1 REM ......................
.......
  10 LET A=16514
  20 LET A$="2A0C40061F233617160
91E501D20FD1520F82336172B360010E
BC9"
  30 FOR B=1 TO LEN A$-1 STEP 2
  40 LET C=CODE A$(B)-28
```



```
 50 LET D=CODE A$(B+1)-28
 60 POKE A,16*C+D
 70 LET A=A+1
 80 NEXT B
100 PRINT AT 0,31;"█"
110 RAND USR 16514
120 GOTO 100
```


Lorsque ce programme est lancé on obtient sur l'écran le même affichage, une étoile filante, qu'avec le programme Basic précédent, mais cette fois il est possible d'augmenter et de diminuer la vitesse de l'étoile, en changeant les données des temporisations 1 et 2 du programme machine.

Après le premier lancement de ce programme, le programme machine est venu se loger dans l'instruction REM de la ligne 1 :

```
   1 REM E£RND▪37Q*-▒2?14 CLEAR
+4 SAVE 7Q*FQ ( FOR TAN ...
  10 LET A=16514
  20 LET A$="2A0C40061F23361716091E501D20FD1520F82336172B360010EBC9"
  30 FOR B=1 TO LEN A$-1 STEP 2
  40 LET C=CODE A$(B)-28
  50 LET D=CODE A$(B+1)-28
  60 POKE A,16*C+D
  70 LET A=A+1
  80 NEXT B
 100 PRINT AT 0,31;"█"
 110 RAND USR 16514
 120 GOTO 100
```

Puisqu'on avait tapé 30 points, il en reste 3 !

Nous pouvons alors supprimer les lignes 10 à 80 du programme devenues inutiles ; il reste alors un programme de 4 lignes qui anime toujours la même étoile filante :

```
   1 REM E£RND▪37Q*-▒2?14 CLEAR
+4 SAVE 7Q*FQ ( FOR TAN ...
 100 PRINT AT 0,31;"█"
 110 RAND USR 16514
 120 GOTO 100
```

Dans ce programme réduit nous pouvons toujours modifier la vitesse de l'étoile à l'aide de la fonction EDIT, mais cela est encore plus facile avec l'instruction POKE.

L'exemple suivant montre le même programme auquel la ligne 90 a été ajoutée afin de permettre la modification de la vitesse de l'étoile en jouant sur la temporisation 1 du programme machine.

En augmentant le nombre 20 de la ligne 90 la vitesse de l'étoile diminue et en diminuant ce nombre la vitesse augmente.

Le demi damier du programme précédent a été remplacé par le signe =

```
   1 REM E£RND▖▘37Q*-=2?14 CLEAR
+4 SAVE 7Q*FQ ( FOR TAN ...
  90 POKE 16523,20
 100 PRINT AT 0,31;"█"
 110 RAND USR 16514
 120 GOTO 100
```

C'est cette ligne qui en est responsable

Dans ce programme machine on remarquera l'avant-dernière instruction, DJNZ, que nous utilisons pour la première fois. Dans ce cas, cette instruction décrémente le registre B et saute 21 cases mémoires en arrière si l'octet contenu dans B n'est pas zéro, c'est-à-dire si l'étoile n'a pas atteint le carré noir.

La méthode que nous avions vue jusqu'à présent nécessitait deux instructions pour obtenir le même résultat.

10 — DÉCALAGE DE L'AFFICHAGE VERS LE BAS

27 PROGRAMME DE DÉCALAGE A (16 K)

Le programme qui suit nécessite l'utilisation du module 16 K de mémoire RAM.

Nous savons que le langage machine permet de créer des fonctions qui n'existent pas dans le Basic du micro-ordinateur ZX 81. Nous allons en voir un exemple avec le programme qui suit. Il va nous permettre de créer une fonction semblable au SCROLL, mais au lieu de décaler l'affichage vers le haut de celui-ci sera décalé vers le bas.

Ce programme sera particulièrement intéressant car il va nous permettre de voir l'utilisation de nouvelles instructions du langage machine du microprocesseur Z 80, et de nombreuses variantes de l'instruction LD. Ces nouvelles instructions sont PUSH et POP qui s'occupent de la pile.

La pile est une partie de la mémoire RAM où des octets peuvent être placés et repris suivant les besoins du programme. L'avantage d'utiliser la pile plutôt que des cases quelconques de la mémoire pour placer des informations en attente est qu'il n'est pas nécessaire de spécifier d'adresse particulière, la pile étant gérée par un registre spécial « pointeur de pile », le registre SP.

L'instruction en langage machine PUSH HL place le contenu de la paire de registres HL au sommet de la pile. L'instruction POP HL place dans la paire de registre HL la dernière information que l'on a placée dans la pile.

Comme le programme qui suit exige la disponibilité de la totalité du fichier d'affichage, il nous faudra utiliser le module d'extension mémoire.

Hexadécimal	**Assembleur**	
2A,0C,40	LD HL, 40 0C	*Adresse du fichier d'affichage dans HL*
11,72,02	LD DE, 0272	*Nombre d'octets à déplacer (626)*
19	ADD HL,DE	*Addition des registres HL et DE*
E5	PUSH HL	*Mise en mémoire de HL dans la pile*
06,21	LD B,21	*Charge dans B le nombre d'octets d'une ligne*
23	INC HL	*Incrémente HL*

10,FD	DJNZ,FD	Décrémente B et saute à − 3 si non nul
E5	PUSH HL	Mise en mémoire de HL dans la pile
D1	POP DE	DE contient les informations de HL
E1	POP HL	Remet HL à sa valeur précédente
0E,13	LD C,13	N° de la ligne (19)
→ 06,21	LD B,21	Charge dans B le nombre 33
→ 7E	LD A, HL	Charge dans A le caractère pointé par HL
12	LD DE ,A	Charge à l'adresse pointée par DE l'octet de A
1B	DEC DE	La paire DE pointe l'adresse suivante
2B	DEC HL	La paire HL pointe l'adresse suivante
10,FA	DJNZ,FA	Répète 33 fois les opérations précédentes
0D	DEC C	Décrémente le N° de la ligne
20,F5	JR NZ,F5	Saut à − 11 si le N° n'est pas 0
C9	RET	Retour au Basic

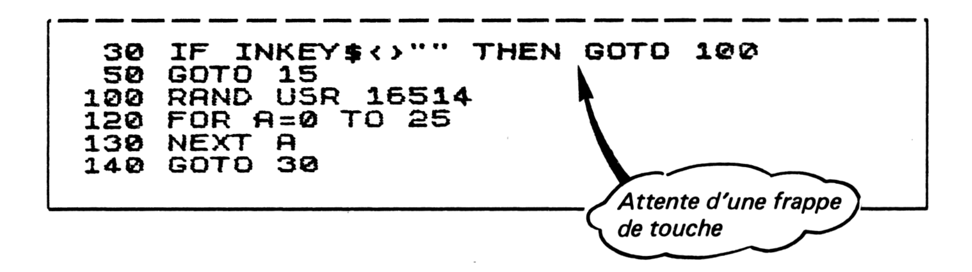
Voici le programme Basic chargeur :

```
    1 REM .........................
.............
   10 LET A=16514
   20 LET A$="2A0C40117202l9E5062
12310FDE5D1E10E1306217E121B2B10F
A0D20F5C9"
   30 FOR B=1 TO LEN A$-1 STEP 2
   40 LET C=CODE A$(B)-28
   50 LET D=CODE A$(B+1)-28
   60 POKE A,16*C+D
   70 LET A=A+1
   80 NEXT B
  100 RAND USR 16514
```


Lorsque ce programme est lancé il ne se passe rien sur l'écran qui reste vide, mais l'instruction REM de la ligne permet de supprimer les lignes 10 à 80 du programme.

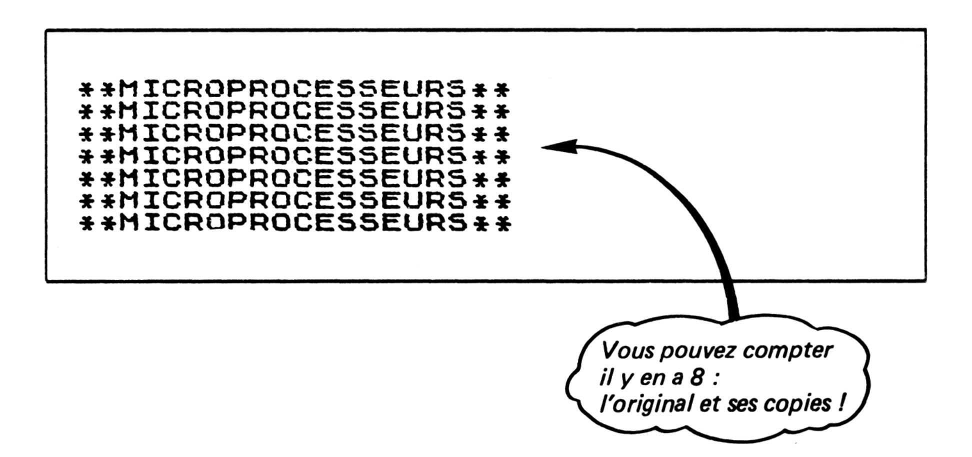
Nous pouvons remplacer ces lignes par d'autres de manière à constituer un programme Basic qui pourra utiliser notre sous-programme machine ainsi que le montre l'exemple suivant :


```
    1 REM E£RND)?▀; FAST ▟57( CLE
AR FAST SGN LPRINT :(▟5$4 PRINT
TAN .......
   10 PRINT AT 0,5;"**MICROPROCES
SEURS**"
```



```
 30 IF INKEY$<>"" THEN GOTO 100
 50 GOTO 15
100 RAND USR 16514
120 FOR A=0 TO 25
130 NEXT A
140 GOTO 30
```


Lorsque ce programme est lancé l'on voit s'inscrire sur l'écran le mot contenu dans la chaîne de caractères, de la ligne 10, dans le cas présent le mot microprocesseur. Chaque fois qu'une touche quelconque est pressée le même texte se répète à la ligne inférieure. L'exemple suivant montre l'affichage sur l'écran après 6 pressions successives sur la touche NEW LINE.



```
**MICROPROCESSEURS**
**MICROPROCESSEURS**
**MICROPROCESSEURS**
**MICROPROCESSEURS**
**MICROPROCESSEURS**
**MICROPROCESSEURS**
**MICROPROCESSEURS**
```

28 PROGRAMME DE DÉCALAGE B (16 K)

Tout comme il est possible d'écrire de plusieurs manières un même programme en langage Basic, il est possible d'écrire un même programme en langage machine de plusieurs façons différentes.

Nous allons donc refaire le programme machine précédent en l'écrivant d'une façon différente.

Cette nouvelle version sera plus courte car nous utilisons des instructions machine différentes. Nous n'utiliserons pas la pile mais des fonctions identiques seront assurées par l'utilisation des trois paires de registres BC,DE,HL.

Dans cette nouvelle version nous utiliserons une instruction très puissante qui permet de transférer des blocs entiers de données d'une adresse à une autre c'est l'instruction LDDR. Les registres H et L doivent contenir l'adresse finale du bloc données source, les registres D et E l'adresse finale du bloc de données destination, les registres B et C le nombre d'octets à transférer.

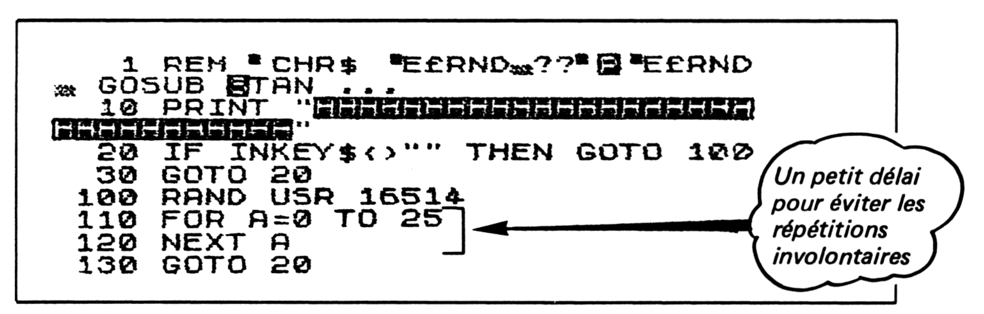
Voici ce nouveau programme machine :

Hexadécimal	**Assembleur**	
01,D6,02	LD BC,02 D6	*Nombre total d'octets du fichier (726)*
2A,0C,40	LD HL, 40 0C	*L'adresse de début du fichier d'affichage*
09	ADD HL,BC	*Addition de HL et BC dans HL*
54	LD D,H	*Transfère H dans D*
5D	LD E,L	*Transfère L dans E*
01,B5,02	LD BC,02 B5	*Fichier moins une ligne (693 octets)*
2A,0C,40	LD HL, 04 0C	*Début du fichier dans HL*
09	ADD HL,BC	*Addition de HL et BC dans HL*
ED,B8	LDDR	*Transfert du bloc de données*
C9	RET	*Retour au Basic*

Programme chargeur Basic :

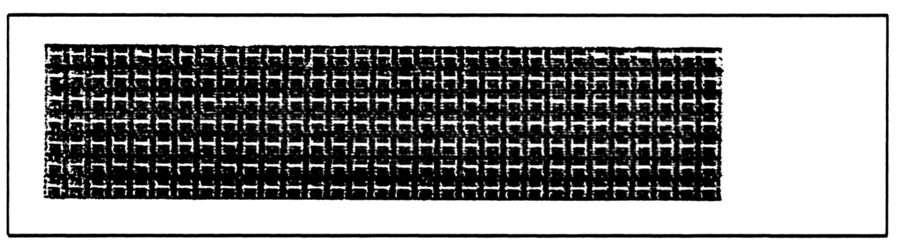
```
   1 REM ........................
  10 LET A=16514
  20 LET A$="01D6022A0C4009545D0
1B5022A0C4009EDB8C9"
  30 FOR B=1 TO LEN A$-1 STEP 2
  40 LET C=CODE A$(B)-28
  50 LET D=CODE A$(B+1)-28
  60 POKE A,16*C+D
  70 LET A=A+1
  80 NEXT B
 100 RAND USR 16514
```

Lorsque ce programme est lancé pour la première fois, le programme machine passe dans l'instruction de la ligne 1. On peut alors effacer les lignes devenues inutiles pour les remplacer par un programme d'application :



La chaîne de caractères de la ligne 10 s'affiche sur l'écran lorsque ce programme est lancé ; cette ligne est répétée un cran plus bas à chaque pression sur une touche.

L'exemple suivant montre l'affichage sur l'écran après six pressions sur la touche NEW LINE.



Pour interrompre le programme, il suffit de presser la touche BREAK.

Ce programme peut être facilement modifié de différentes manières. Il est possible d'effacer la ligne précédente en même temps que l'on remplit la ligne inférieure. Il est possible de combiner ce sous-programme machine avec l'instruction SCROLL de manière à remonter d'une ligne le texte affiché sur l'écran en pressant sur une touche et de faire descendre ce texte d'une ligne en pressant une autre touche.

11 — LES NOMBRES NÉGATIFS

29 SOUSTRACTION AVEC RÉSULTAT NÉGATIF (1 K)

Nous savons qu'un octet peut correspondre à un nombre positif de 0 à 255 (ou 00 à FF en hexadécimal), mais un octet peut également représenter un nombre algébrique positif ou négatif.

Dans les programmes machine que nous avons vu précédemment nous avons souvent utilisé des octets négatifs comme adresse dans des instructions de saut.

Pour comprendre le signe algébrique d'un octet il faut se rappeler que lorsqu'un octet est à 00 et qu'on ajoute 1 il passe à 01, mais si on retranche 1 il passe à 00 en hexadécimal ou 255 en décimal. C'est pourquoi FF correspond à −1, FE à −2 etc...

Le programme machine suivant montre une soustraction dont le résultat est négatif : 6−9.

Voici ce programme en langage machine :

Hexadécimal	**Assembleur**	
3E,06	LD A,06	*Chargement de 06 dans A*
0E,09	LD C,09	*Chargement de 09 dans C*
91	SUB C	*Soustraction de C dans A*
06,00	LD B,00	*Chargement de 00 dans B*
4F	LD C,A	*Chargement de A dans C*
C9	RET	

L'exemple suivant montre ce programme machine placé dans la chaîne de caractères A$ à la ligne 20 du programme chargeur.

Lorsque ce programme est lancé, on voit 253 s'afficher sur l'écran ; ce nombre correspond à FD en hexadécimal donc à −3, le résultat attendu, mais on n'a pas tenu compte de la retenue négative.

```
  1 REM .............
 10 LET A=16514
 20 LET A$="3E060E099106004FC9"
 30 FOR B=1 TO LEN A$-1 STEP 2
 40 LET C=CODE A$(B)-28
 50 LET D=CODE A$(B+1)-28
 60 POKE A,16*C+D
 70 LET A=A+1
 80 NEXT B
100 PRINT USR 16514
```

Lorsque ce programme est lancé pour la première fois, le programme machine vient se placer dans l'instruction REM de la ligne 1, ce qui permet de supprimer les lignes 10 à 80 qui sont devenues inutiles.

```
  1 REM Y▀:▒█▀ ?TAN ....
100 PRINT USR 16514
```

Même sous cette forme réduite il est possible de modifier le programme machine, à l'aide de la fonction EDIT. Dans l'exemple qui suit nous avons modifié le programme machine en remplaçant le deuxième caractère du programme machine qui équivalait à l'octet 06 (▛) par un autre caractère qui équivaut à l'octet 01 (graphics 1).

```
  1 REM Y▗▀:▒█▀ ?TAN ....
100 PRINT USR 16514
```

Lorsque ce programme modifié est lancé, Le résultat de l'opération qui s'affiche sur l'écran n'est plus 253 mais 248 ou F8 c'est-à-dire – 8 qui est le résultat de la soustraction 1 moins 9.



AFFICHAGE DES OCTETS POSITIFS ET NÉGATIFS (1 K)

Nous allons voir un autre programme machine qui va mettre en évidence le signe des dix premiers octets positifs et des 10 premiers octets négatifs.

Ce programme machine charge le nombre 00 dans le registre A puis place le contenu du registre A dans la case mémoire dont l'adresse est 17152, avant de retourner au programme Basic.

Voici le programme machine :

Hexadécimal	**Assembleur**	
3E,00	LD A,00	*Charger 00 dans A*
32,00,43	LD 43 00 ,A	*Charger A à l'adresse 17152*
C9	RET	*Retour au Basic*

Le programme Basic chargeur :

```
  1 REM ........
 10 LET A=16514
 20 LET A$="3E0032004 3C9"
 30 FOR B=1 TO LEN A$-1 STEP 2
 40 LET C=CODE A$(B)-28
 50 LET D=CODE A$(B+1)-28
 60 POKE A,16*C+D
 70 LET A=A+1
 80 NEXT B
```

Lorsque ce programme est lancé pour la première fois, le programme machine passe dans l'instruction REM de la ligne 1 en remplacement des points placés à cette ligne et donne le programme machine réduit.

Ce programme machine va servir de sous-programme au programme en langage Basic qui suit.

Ce programme imprime sur l'écran les nombres algébriques de + 10 à – 10 et à côté de chacun de ces nombres sera affiché l'octet correspondant.

```
  1 REM Y PLOT M ?TAN ..
 10 FOR X=10 TO -10 STEP -1
 20 POKE 16515,X
 30 RAND USR 16514
 40 PRINT X;"....";PEEK 17152
 50 NEXT X
```

L'exemple suivant montre l'affichage qui s'inscrit sur l'écran lorsque ce programme est lancé. On remarquera la transition qui se produit lorsque l'octet devient négatif.

```
10....10
9....9
8....8
7....7
6....6
5....5
4....4
3....3
2....2
1....1
0....0
-1....255
-2....254
-3....253
-4....252
-5....251
-6....250
-7....249
-8....248
-9....247
-10....246
```

Les octets de 00 à 7F correspondent aux valeurs positives de 0 à 127 en numération décimale.

Les octets de FF à 80 correspondent aux valeurs négatives de − 1 à − 128 en numération décimale.

APPLICATIONS DU LANGAGE MACHINE

31 HORLOGE NUMÉRIQUE (1 K)

Le programme que nous allons voir constitue une application utilitaire d'un sous-programme en langage machine, qui permettra de transformer la version 1 K de notre ZX 81 en une horloge digitale.

Le programme machine va lire à l'adresse 16436 de la mémoire RAM le compteur des trames affichées sur l'écran, ce compteur est utilisé pour les instructions PAUSE de temporisation ; il compte à la fréquence du secteur c'est-à-dire cinquante fois par seconde.

Notre programme en langage machine compte 250 impulsions avant de provoquer le changement de l'affichage de l'horloge. L'affichage de l'horloge est donc modifié toutes les 5 secondes. Voici ce programme machine :

Hexadécimal	Assembleur	
00	00	*Variable*
3A,82,40	LD A, 40 82	*Charger l'octet N° 16514 dans A*
D6,FA	SUB A,FA	*Soustraction : A moins 250*
32,82,40	LD 4082 ,A	*Charger A dans l'octet N° 16514*
→21,34,40	LD HL,4034	*Charger HL avec le contenu de 16436*
56	LD D, HL	*Charger dans D l'octet pointé par HL*
BA	CP D	*Comparer A et D*
20,FC	JRNZ,FC	*Saut à −4 si le résultat est différent de zéro*
C9	RET	*Retour au Basic*

Le programme Basic chargeur :

```
   1 REM .......................
...
  10 LET A=16514
  20 LET A$="003A8240D6FA3282402
1344056BA20FCC9"
  30 FOR B=1 TO LEN A$-1 STEP 2
  40 LET C=CODE A$(B)-28
  50 LET D=CODE A$(B+1)-28
  60 POKE A,16*C+D
  70 LET A=A+1
  80 NEXT B
 100 RAND USR 16514
```

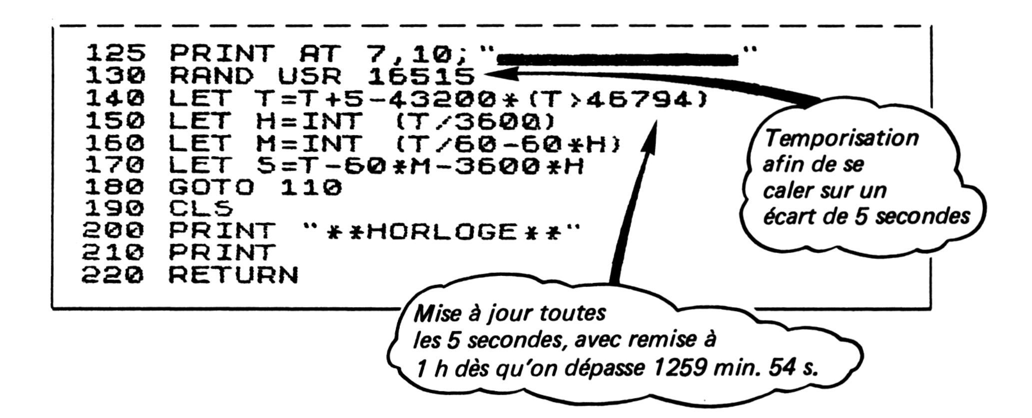
Lorsque ce programme est lancé, le programme machine vient se placer dans l'instruction REM de la ligne 1 en remplacement des points de cette ligne, ce qui permet de supprimer les autres lignes de ce programme devenues inutiles pour ne conserver que la ligne 1 qui servira de sous-programme à notre programme Basic d'horloge.

L'exemple suivant montre le programme complet de l'horloge digitale avec le sous-programme en langage machine contenu dans l'instruction REM de la ligne 1.

```
   1 REM ?U▙RNDCHR$ IF M▙RND5ORN
D?▣4 UNPLOT TAN .........
   5 GOSUB 190
  15 PRINT "REGLAGE HEURES"
  20 INPUT H
  35 PRINT
  40 PRINT "REGLAGE MINUTES"
  50 INPUT M
  60 LET S=0
  70 POKE 16514,80-256*(PEEK 164
36>175)+PEEK 16436
  80 IF S<0 OR S>59 OR H<0 OR H>
12 OR M<0 OR M>59 THEN GOTO 5
  90 LET T=S+3600*H+60*M
 100 GOSUB 190
 105 PRINT AT 5,10;"▀▀▀▀▀▀▀▀▀▀▀▀"
 110 PRINT AT 6,10;"............"
 120 PRINT AT 6,11;H;"..";M;".."
S
- - - - - - - - - - - - - - - - -
```

Amorce pour l'utilisation du sous-programme machine

L'heure exprimée en secondes



```
125 PRINT AT 7,10;"████████████████"
130 RAND USR 16515
140 LET T=T+5-43200*(T>46794)
150 LET H=INT (T/3600)
160 LET M=INT (T/60-60*H)
170 LET S=T-60*M-3600*H
180 GOTO 110
190 CLS
200 PRINT "**HORLOGE**"
210 PRINT
220 RETURN
```


La partie allant de la ligne 5 à la ligne 90 de ce programme est relative à la mise à l'heure de l'horloge digitale.

Lorsque ce programme est lancé le ZX 81 vous demande le réglage de l'heure par exemple 4, ensuite il demande le réglage des minutes, vous mettrez environ une minute de plus que l'heure exacte par exemple 15. Lorsque l'heure exacte que vous venez de régler est atteinte, vous pressez la touche NEW LINE et le programme d'horloge commence.

L'affichage sur l'écran est montré dans l'exemple qui suit. Toutes les 5 secondes cet affichage change pour indiquer l'heure exacte, du fait de la ligne 130.

```
**HORLOGE**



                ████████████████
                .4..15..20.
                ████████████████
```

Lorsque le programme d'horloge est lancé, c'est la partie allant de la ligne 110 à la ligne 180 du programme qui est constamment en service avec le programme machine.

La ligne 140 du programme est particulière : c'est elle qui incrémente de 5 secondes l'heure affichée sur l'écran à chaque retour du sous-programme machine ; de plus cette ligne provoque l'indication : *1 heure, 0 minute, 0 seconde* lorsque l'horloge atteint 13 h.

LES INSTRUCTIONS LOGIQUES DU MICROPROCESSEUR Z 80

Les instructions logiques du microprocesseur Z 80 s'utilisent de la même manière que les instructions d'addition ou de soustraction que nous avons eu l'occasion d'utiliser dans les programmes précédents.

Lorsqu'on utilise un micro-ordinateur pour des applications industrielles, les instructions logiques sont très importantes pour manipuler les bits d'un octet.

Les ports d'entrée et sortie sont des genres de registres qui permettent à l'unité centrale d'un ordinateur de communiquer avec des circuits extérieurs.

Les octets placés dans ces ports peuvent servir à commander des circuits électriques afin de les automatiser, à déterminer si une grandeur électrique est à son réglage optimum. Cette grandeur peut représenter la valeur d'une température, d'une vitesse, d'une pression etc... Dans le cas où cette valeur s'écarte de la normale, le programme pourra agir sur des commandes pour retrouver automatiquement le réglage correct.

Nous allons donc voir quelques courts programmes machine ayant pour but la manipulation des octets à l'aide des instructions logiques.

Une instruction logique, c'est une comparaison entre l'octet contenu dans le registre accumulateur et un autre octet. Le résultat de l'opération se retrouve dans le registre accumulateur. Pour comprendre les opérations logiques dont il est question dans ce chapitre, il est indispensable de reprendre les explications données précédemment sur la numération binaire.

32 L'INSTRUCTION AND (1 K)

Le premier programme que nous allons voir concernera l'instruction AND : lorsque cette instruction est utilisée dans un programme machine, chacun des bits de l'octet contenu dans le registre accumulateur est comparé avec les bits correspondants du deuxième octet. Le bit reste à 1 si, dans les deux octets, le bit correspondant est à 1 ; dans les autres cas, ce bit passe à 0.

Prenons par exemple deux octets, le premier est 115 en décimal ou AF en hexadécimal qui s'écrit 1010 1111 en binaire, le deuxième octet est 119 en décimal ou 77 en hexadécimal qui s'écrit 0111 0111 en binaire.

En réalisant une opération AND entre ces deux octets nous obtenons 39 en décimal comme résultat ou 27 en hexadécimal soit 00100111 en binaire.

Les opérations AND sont souvent utilisées pour masquer certains des bits d'un octet. Par exemple si nous ne voulons pas utiliser les bits 4, 5, 6, 7 d'un octet nous effectuons une opération AND avec l'octet 0F en hexadécimal : tous les bits de poids supérieurs à ce nombre seront éliminés dans l'octet contenu dans le registre A.

Le programme machine qui suit va montrer une application de cette utilisation de l'instruction AND.

	Hexadécimal	**Assembleur**	
16514	3E,00	LD A,0	*Charger 00 dans A*
	32,00,43	LD 4300 ,A	*Placer l'octet de A à 17152*
	32,02,43	LD 4302 ,A	*Placer l'octet de A à 17154*
	C9	RET	*Retour au Basic*
16523	06,0F	LD B,0F	*Charger 0F dans B*
	3A,00,43	LD A, 4300	*Placer l'octet de 17152 dans A*
	3C	INC,A	*Incrémentation de A*
	32,00,43	LD 4300 ,A	*Placer A à 17152*
	A0	AND B	*Opération AND entre B et A*
	32,02,43	LD 4302 ,A	*Placer A à 17154*
	C9	RET	*Retour au Basic*

Ce programme machine comprend deux parties, la première met à zéro le contenu des cases 17152 et 17154 de la mémoire RAM.

La deuxième partie place dans le registre B l'octet 0F (15 en décimal), ensuite le contenu de l'adresse 17152 est chargé dans A, ensuite A est incrémenté et son contenu est replacé à l'adresse 17152, puis A subit une opération AND avec le registre B enfin le contenu de A est placé à l'adresse 17154.

Voici le programme chargeur :

Il y a ici deux sous-programmes machine !

```
   1 REM ........................
..
  10 LET A=16514
  20 LET A$="3E0032004332024 3C90
60F3A00433C320043A032024 3C9"
  30 FOR B=1 TO LEN A$-1 STEP 2
  40 LET C=CODE A$(B)-28
  50 LET D=CODE A$(B+1)-28
  60 POKE A,16*C+D
  70 LET A=A+1
  80 NEXT B
```

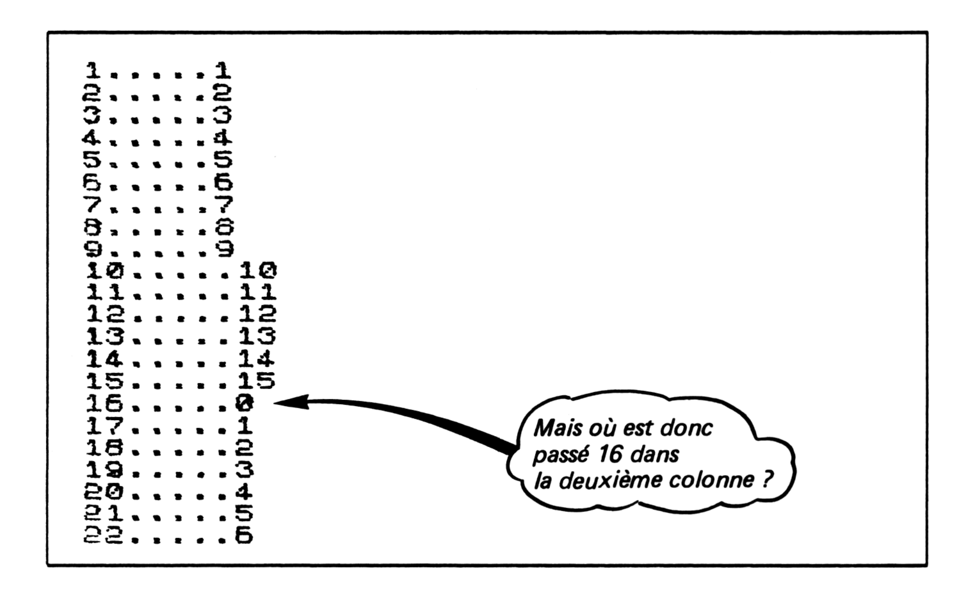
Lorsque ce programme est lancé, les deux sous-programmes machine passent dans l'instruction REM de la ligne 1 ce qui permet de supprimer les lignes 10 à 80 du programme qui sont devenues inutiles.

Nous ajoutons ensuite les lignes en Basic destinées à exécuter le premier sous-programme machine et ensuite à utiliser 22 fois le deuxième c'est-à-dire à afficher 22 fois sur l'écran le contenu de l'adresse 17152 qui est incrémenté à chaque fois, et également le contenu de l'adresse 17154 qui subit en plus une opération logique AND avec l'octet 15 contenu dans le registre B, ce qui interdit au contenu de cette adresse de dépasser la valeur 15 (0F en hexadécimal).

L'exemple suivant montre le programme Basic avec les sous-programmes machine contenus dans l'instruction REM de la ligne 1.

```
   1 REM Y M ?M ▀?TAN ▟?U ?WM ?▣H
▀?TAN ..
 100 RAND USR 16514
 110 FOR X=0 TO 21
 120 RAND USR 16523
 130 PRINT PEEK 17152;".....";PE
EK 17154
 140 NEXT X
```

Et voici ce qu'affiche l'écran lorsque ce programme est lancé :



La première colonne montre le contenu de l'adresse 17152 qui est constamment incrémenté, la deuxième colonne montre le contenu de l'adresse 17154 qui subit en plus une opération AND qui masque tous les octets supérieurs à 15. C'est pourquoi le contenu de cette adresse revient à zéro au lieu d'afficher 16. En effet 16 correspond au bit 4 qui est masqué.

33 L'INSTRUCTION OR (1 K)

Nous allons voir une autre opération logique avec l'instruction OR. Lorsque l'on effectue une opération OR (OU en français) sur deux bits ; on obtient 1 comme résultat si au moins un des deux bits est à 1, et le résultat est zéro seulement dans le cas où les deux bits sont à zéro.

Prenons les 2 octets de l'exemple précédent qui sont AF et 77 en hexadécimal soit 10101111 et 01110111 en binaire, si sur ces deux octets, nous effectuons une opération OR le résultat sera FF (255 en décimal), ce qui s'écrit 11111111 en numération binaire.

Les opérations logiques OR servent souvent à mettre à la valeur 1 certains des bits d'un octet, comme nous allons le voir dans le programme d'application qui suit où l'octet contenu à l'adresse 17 154 va subir une opération logique OR avec l'octet 01 contenu dans le registre B.

Les conséquences de cette opération logique OR feront que la valeur de cet octet sera toujours impaire.

Voici ce programme machine qui ressemble beaucoup au précédent :

	Hexadécimal	**Assembleur**	
16514	3E,00	LD A,00	*Charge 00 dans A*
	32,00,43	LD 4300 ,A	*Place A à 17152*
	32,02,43	LD 4302 ,A	*Place A à 17154*
	C9	RET	*Retour au Basic*
16523	06,01	LD B,01	*Charge 01 dans B*
	3A,00,43	LD A, 4300	*Charge A avec l'octet de 17152*
	3C	INC A	*Incrémente A*
	32,00,43	LD 4300 ,A	*Place A à 17152*
	B0	OR B	*Opération OR entre A et B*
	32,02,43	LD 4302 ,A	*Place A à 17154*
	C9	RET	*Retour au Basic*

L'exemple qui suit montre ce programme en langage machine placé dans la chaîne de caractères A$ à la ligne 20 du programme chargeur lorsque ce programme est prêt à être lancé.

Une fois ce programme lancé le programme machine passera dans l'instruction REM de la ligne 1 en remplacement des points, ce qui permettra de supprimer les lignes 10 à 80 du programme chargeur devenues alors inutiles.

```
   1 REM Y M ?M ?TAN  U ?WM ?H
 ?TAN ..
  10 LET A=16514
  20 LET A$="3E0032004332024 3C90
6013A00433C320043B032024 3C9"
  30 FOR B=1 TO LEN A$-1 STEP 2
  40 LET C=CODE A$(B)-28
  50 LET D=CODE A$(B+1)-28
  60 POKE A,16*C+D
  70 LET A=A+1
  80 NEXT B
```

Le programme machine contenu dans l'instruction REM de la ligne 1 sert de sous-programmes au programme Basic suivant. Le premier sous-programme en langage machine commence à l'adresse 16514 et il est appelé par la ligne 100. Ensuite c'est le deuxième sous-programme, situé à partir de l'adresse 16523 qui est exécuté 22 fois grâce à la ligne 120 :

```
   1 REM Y M ?M ?TAN  U ?WM ?H
 ?TAN ..
 100 RAND USR 16514
 110 FOR X=0 TO 21
 120 RAND USR 16523
 130 PRINT PEEK 17152;".....";PE
EK 17154
 140 NEXT X
```

Lorsque ce programme est lancé le contenu des adresses 17152 et 17 154 est affiché 22 fois sur l'écran, comme le montre l'exemple suivant.

```
1.....1
2.....3
3.....3
4.....5
5.....5
6.....7
7.....7
8.....9
9.....9
10.....11
11.....11
12.....13
```

```
13.....13
14.....15
15.....15
16.....17
17.....17
18.....19
19.....19
20.....21
21.....21
22.....23
```

Le contenu des deux adresses 17152 et 17154 de la mémoire RAM est régulièrement incrémenté d'une unité à chaque cycle mais, comme l'octet placé à l'adresse 17154 subit l'opération logique OR avec 1, l'octet qui est à cette adresse ne peut être qu'un nombre impair.

Si nous avions établi un programme semblable avec une opération logique AND et la donnée 254 ou FE en hexadécimal dans le registre B le contenu de l'adresse 17154 aurait toujours été un nombre pair (bit de droite toujours nul).

34 L'INSTRUCTION XOR (1 K)

Nous allons voir une dernière opération logique qui utilise l'instruction XOR (Exclusive OR, OU exclusif). Cette instruction donne un résultat semblable à OR avec cette différence que si les deux bits faisant l'objet de l'opération XOR sont à 1, le résultat est zéro. Un OU exclusif sur deux octets identiques donne donc zéro.

Si l'on exécute une opération XOR sur les deux octets AF et 77 en hexadécimal qui nous ont servi d'exemple précédemment, le résultat sera D8 en hexadécimal, 216 en décimal, ou encore 11011000 en binaire.

On peut exécuter une opération XOR sur l'accumulateur A lui-même. Cela revient à mettre à 0 tous les bits de l'accumulateur, ce qui emploie une instruction de un octet alors que l'instruction LDA,00 que nous utilisons habituellement demande deux octets.

Nous pouvons utiliser l'instruction XOR pour modifier un octet placé dans une case mémoire, ainsi que le montre le programme qui va suivre.

Dans ce programme machine, nous allons charger l'octet 00 à l'adresse 17152, puis nous reprenons cet octet pour lui faire subir une opération XOR avec l'octet 01 contenu dans le registre B ; lorsque cet octet est à 0 il passe à 1 et s'il est à 1 il passe à 0 en s'inversant à chaque lancement.

Voici ce programme en langage machine.

	Hexadécimal	**Assembleur**	
16514	3E,00	LD A,00	*Charge 00 dans A*
	32,00,43	LD 4300 ,A	*Charge A à l'adresse 17152*
	C9	RET	*Retour au Basic*
	00	NOP	*Sans opération*
16521	06,01	LD B,01	*Charge 01 dans B*
	3A,00,43	LD A, 4300	*Charge A avec l'octet de 17152*
	A8	XOR B	*Opération XOR entre B et A*
	32,00,43	LD 4300 ,A	*Charge A à l'adresse 17152*
	C9	RET	*Retour au Basic*

Voici le programme chargeur :

```
   1 REM ......................
  10 LET A=16514
  20 LET A$="3E003200430C900006013
A00043A83200043C9"
  30 FOR B=1 TO LEN A$-1 STEP 2
  40 LET C=CODE A$(B)-28
  50 LET D=CODE A$(B+1)-28
  60 POKE A,16*C+D
  70 LET A=A+1
  80 NEXT B
```

Lorsque ce programme est lancé, le programme en langage machine vient se placer dans l'instruction REM de la ligne 1 à la place des points. Les lignes 10 à 80 devenues inutiles peuvent être supprimées.

Le programme Basic qui suit utilise le sous-programme en langage machine employant l'instruction XOR.

```
   1 REM Y M ?TAN  U ?M ?TAN
......
 100 RAND USR 16514
 110 FOR X=0 TO 44
 120 RAND USR 16520
 130 PRINT PEEK 17152;"...";
 140 NEXT X
```

Lorsque ce programme est lancé, on voit le contenu de l'adresse 17152 s'afficher 45 fois sur l'écran, mais comme à chaque fois l'octet contenu à cette adresse subit une opération XOR avec l'octet 01 contenu dans le registre B, l'octet passe à 1 s'il était à 0 et à 0 s'il était à 1 :

```
1...0...1...0...1...0...1...0...
1...0...1...0...1...0...1...0...
1...0...1...0...1...0...1...0...
1...0...1...0...1...0...1...0...
1...0...1...0...1...0...1...0...
1...0...1...0...1...
```

35 VA ET VIENT (1 K)

Le programme que nous allons voir est une autre façon d'écrire un programme donné en Basic dans le chapitre sur les mouvements rapides de l'affichage. Ce programme montrait une étoile qui exécute un mouvement de va et vient entre deux carrés en haut de l'écran. Nous allons l'écrire en langage machine :

	Hexadécimal	**Assembleur**	
16514	2A,0C,40	LD HL, 40 0C	*Adresse du fichier d'affichage dans HL*
	06,1D	LD B,1D	*Charge dans B le nombre de pas*
	23	INC HL	*Un pas à droite*
	→23	INC HL	*Un pas à droite*
	36,17	LD HL ,17	*Affiche une étoile*
	CD,A7,40	CALL 40,A7	*Appel sous-programme à 16551*
	23	INC HL	*Un pas à droite*
	36,17	LD HL ,17	*Affiche une étoile*
	2B	DEC HL	*Un pas à gauche*
	36,00	LD HL ,00	*Efface l'étoile précédente*
	10,F2	DJNZ,F2	*Décrément et saut à – 14 si B n'est pas 0*
	06,1D	LD B,1D	*Charge dans B le nombre de pas*
	23	INC HL	*Un pas à droite*
	→2B	DEC HL	*Un pas à gauche*
	36,17	LD HL ,17	*Affiche une étoile*
	23	INC HL	*Un pas à droite*
	36,00	LD HL ,00	*Efface l'étoile précédente*
	2B	DEC HL	*Un pas à gauche*
	CD,A7,40	CALL 40,A7	*Appel sous-programme à 16551*
	10,F4	DJNZ,F4	*Décrément et saut à – 12 si B n'est pas 0*
	C9	RET	*Retour au Basic*
	00	NOP	*Opération nulle*
16551	00	NOP	*Opération nulle*
	16,0F	LD D,OF	*Charge 15 dans D*
	→IE,FF	LD E,FF	*Charge 255 dans E*
	→1D	DEC E	*Décrémente E*
	20,FD	JR NZ,FD	*Si E n'est pas 0 saut à – 3*
	15	DEC D	*Décrémente D*
	20,F8	JR NZ,F8	*Si D n'est pas 0 saut à – 8*
	C9	RET	*Retour au programme principal*

Ce programme machine est intéressant pour deux raisons :

Premièrement nous voyons comment un sous-programme machine, comme le sous-programme de temporisation des mouvements de l'étoile sur l'écran peut être introduit dans le programme machine principal.

Deuxièmement lorsqu'un programme machine a une certaine longueur il devient difficile de placer le programme entier dans une chaîne de caractères comme nous l'avons fait jusqu'à présent.

Nous allons voir comment un programme machine peut être chargé progressivement dans l'instruction REM du programme chargeur.

L'exemple suivant montre le programme chargeur avec la première partie du programme machine contenue dans la chaîne de caractères A$ de la ligne 20.

Vous pouvez les compter : il y a 51 points. Il en faut au moins 49 !

```
    1 REM .........................
..........................
   10 LET A=16514
   20 LET A$="2A0C40061D232336170
DA7402336172B360010F2061D232B361
7"
   30 FOR B=1 TO LEN A$-1 STEP 2
   40 LET C=CODE A$(B)-28
   50 LET D=CODE A$(B+1)-28
   60 POKE A,16*C+D
   70 LET A=A+1
   80 NEXT B
```

Le début du programme machine

Lorsque ce programme est lancé normalement par une commande RUN, la partie du programme machine contenue dans la chaîne de caractères A$ de la ligne 20 se place dans l'instruction REM de la ligne 1 en remplacement des points, ainsi que le montre l'exemple suivant :

```
    1 REM E£RND▘177Q*LN ▓RND7Q*FQ
  ( PAUSE ▘17FQ*..................
........
   10 LET A=16514
   20 LET A$="2A0C40061D232336170
DA7402336172B360010F2061D232B361
7"
   30 FOR B=1 TO LEN A$-1 STEP 2
   40 LET C=CODE A$(B)-28
   50 LET D=CODE A$(B+1)-28
   60 POKE A,16*C+D
   70 LET A=A+1
   80 NEXT B
```

On écrit à nouveau la ligne 20 en y plaçant la suite du programme machine :

```
   1 REM E£RND▪"177Q*LN ▒RND7Q*FQ
  ( PAUSE ▪"17FQ*..................
........
   20 LET A$="2336002BCDA74010F4C
90000160F1EFF1D20FD1520F8C9"
   30 FOR B=1 TO LEN A$-1 STEP 2
   40 LET C=CODE A$(B)-28
   50 LET D=CODE A$(B+1)-28
   60 POKE A,16*C+D
   70 LET A=A+1
   80 NEXT B
```

On a enlevé la ligne 10, inutile

Suite et fin

Non ! ne tapez pas RUN...

Pour lancer ce programme on ne doit pas utiliser RUN ce qui effacerait l'adresse où doit être placé le prochain octet du programme machine. Il faut utiliser GOTO 2Ø qui provoque le positionnement du reste du programme machine à la suite de la première partie dans l'instruction REM de la ligne 1 :

```
   1 REM E£RND▪"177Q*LN ▒RND7Q*FQ
  ( PAUSE ▪"17FQ*7Q FLN ▒RND( POKE
  TAN    -?2 COPY 14 CLEAR +4 SAVE
  TAN ..
   20 LET A$="2336002BCDA74010F4C
90000160F1EFF1D20FD1520F8C9"
   30 FOR B=1 TO LEN A$-1 STEP 2
   40 LET C=CODE A$(B)-28
   50 LET D=CODE A$(B+1)-28
   60 POKE A,16*C+D
   70 LET A=A+1
   80 NEXT B
```

Un programme machine plus long aurait pu être divisé en autant de parties qu'il est jugé nécessaire, chaque partie étant chargée de la même manière par une instruction GOTO 2Ø.

Après le premier lancement du programme de chargement il est possible de supprimer la ligne 10 car l'adresse indiquée par la variable A, où doit être placé le prochain octet du programme machine, n'est plus 16514.

Lorsque le programme machine est entièrement placé dans l'instruction REM du programme chargeur les lignes de ce programme devenues inutiles sont supprimées, et on ne garde que la ligne 1 qui contient le programme machine.

Il n'est possible de lancer ce programme machine directement par un USR 16514 que si votre micro-ordinateur ZX 81 est équipé d'une extension mémoire RAM. Si vous ne disposez que des 1 K de la version de base et que vous lanciez ce programme, votre ZX 81 se bloquerait en essayant d'écrire sur un fichier d'affichage qui n'existe pas. C'est pourquoi, dans l'application qui suit, la ligne 20 de ce programme ouvre la ligne supérieure du fichier d'affichage à l'aide d'une instruction PRINT pour permettre au programme machine de fonctionner même avec 1 K de mémoire RAM.

Le programme d'application qui suit se sert du programme machine placé dans la ligne 1 comme d'un sous-programme.

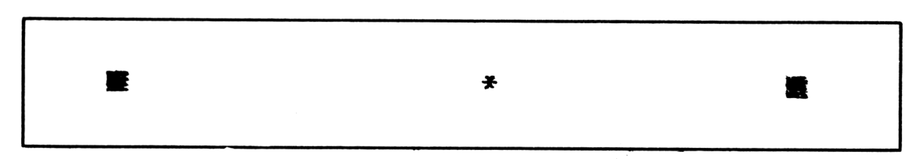
Lorsque ce programme est lancé deux carrés noirs s'affichent dans le haut de l'écran à droite et à gauche et l'on voit une étoile qui fait 10 allers et retours entre ces deux carrés noirs avant que le programme ne se termine.

Voici le programme d'application du programme machine que nous venons de voir :

```
  1 REM E£RND▘1770*LN ▒RND7Q*FQ
( PAUSE ▘17FQ*7Q FLN ▒RND( POKE
TAN    -?2 COPY 14 CLEAR +4 SAVE
TAN ..
 10 FOR A=0 TO 10
 20 PRINT AT 0,0;"■";AT 0,31;"■
"
 30 RAND USR 16514
 40 NEXT A
```

Un aller-retour

Lorsque ce programme est lancé on voit l'étoile qui effectue son mouvement de va et vient entre les deux carrés noirs :



Il est possible de régler la vitesse de déplacement de l'étoile en modifiant les données de la temporisation qui ralentit ses mouvements dans le programme machine : par exemple en remplaçant dans le registre E l'octet FF par l'octet 42.

Avec l'instruction EDIT, il faut changer, dans la ligne 1, COPY par PI.

36 DÉCODAGE DES MESSAGES SECRETS (1 K)

L'ordinateur est l'instrument idéal pour coder et décoder les messages secrets.

Nous allons nous servir du langage machine afin d'établir un programme simple qui transformera un message en clair en un message codé, et inversement.

Ce programme machine est basé sur le code des caractères du micro-ordinateur ZX 81, qui figure dans l'annexe A du manuel Sinclair. La lettre A, par exemple, est représentée par le code 38, soit 26 en hexadécimal.

Dans notre programme machine nous utiliserons les caractères dont les codes sont compris entre 27 et 63 ou entre 1B et 3F en numération hexadécimale. Le code 1B correspondant au point comme caractère, 1C au chiffre 0 et ainsi de suite, jusqu'à 3F qui correspond à la lettre Z.

Le programme machine est centré sur la lettre H dont le code correspond à 45 (2D en hexadécimal). Les caractères dont le code est inférieur ou égal à celui de H verront leur code augmenté de 18 (12 en hexadécimal), les caractères dont le code est supérieur à H verront leur code diminuer de ce même nombre. Le programme machine se divise en deux parties, suivant que le code du caractère examiné est inférieur ou supérieur à celui de la lettre H. Le message primitif sera ainsi transformé pour donner le message codé.

Par la suite il suffit de retraiter le message codé avec le même programme machine pour obtenir à nouveau le message primitif en langage clair.

Voici le programme machine :

Hexadécimal	**Assembleur**	
3A,00,43	LD A, 4300	*Charger l'octet 17152 dans A*
D6,2D	SUB A,2D	*Soustraire 45 de A*
F2,93,40	JPP 4093	*Si A positif saut à 16531 (NOP)*
3A,00,43	LD A, 4300	*Charger l'octet 17152 dans A*
C6,12	ADD A,12	*Ajouter 18 à A*
32,02,43	LD 4302 ,A	*Placer A à 17154*
C9	RET	*Retour au Basic*
→ 00	NOP	*Pas d'opération*
3A,00,43	LD A, 4300	*Charger l'octet 17152 dans A*
D6,12	SUB A,12	*Soustraire 18 de A*
32,02,43	LD 4302 ,A	*Placer A à 17154*
C9	RET	*Retour au Basic*

Précédemment nous avons utilisé des instructions de saut, dans lesquelles le saut était déterminé par la présence ou l'absence de la donnée zéro dans un registre. Dans ce programme machine nous utilisons l'instruction JPP qui ordonne de sauter à l'adresse 16531 — où se trouve placé l'octet 00 qui correspond à l'instruction NOP — dans le cas où le contenu du registre A est positif.

Le programme Basic chargeur :

```
    1 REM .........................
.........
   10 LET A=16514
   20 LET A$="3A0043D62DF293403A0
043C61232024 3C9003A0043D61232024
3C9"
   30 FOR B=1 TO LEN A$-1 STEP 2
   40 LET C=CODE A$(B)-28
   50 LET D=CODE A$(B+1)-28
   60 POKE A,16*C+D
   70 LET A=A+1
   80 NEXT B
  110 RAND USR 16514
```

Lorsque ce programme est lancé pour la première fois, le programme machine vient remplacer les points dans l'instruction REM de la ligne 1. Ceci permet de ne conserver que cette ligne du programme en effaçant les autres devenues inutiles.

Cette ligne 1 du programme chargeur contenant le programme en langage machine va servir de sous-programme au programme codeur de message secret qui suit.

Dans ce programme le message à coder est contenu dans la chaîne de caractères A$ à la ligne 10. Ce message est la phrase « Bonjour quel est votre nom », mais tout autre message serait aussi valable. On a remplacé les espaces (code 0) par des points (code 27) afin de rester dans la gamme de codes convenue (27 à 63).

Voici le programme codeur décodeur de messages.

```
    1 REM U ?CHR$ H PAUSE ▊RNDU ?
LEN >M ▘?TAN  U ?CHR$ >M ▘?TAN ...
...
   10 LET A$="BONJOUR.QUEL.EST.VO
TRE.NOM."
   20 PRINT A$
   30 FOR N=1 TO LEN A$
   40 POKE 17152,CODE A$(N)
   50 RAND USR 16514
   60 LET A=PEEK 17154
   70 PRINT CHR$ A;
   80 NEXT N
```

On fournit la donnée au sous-programme machine

On utilise ce sous-programme

On récupère le résultat

Lorsque ce programme est lancé le message s'affiche en clair sur l'écran du téléviseur, puis le message code s'affiche sous le message en clair :

```
BONJOUR.QUEL.EST.VOTRE.NOM.
T6516C9H8CW3HWABHD6B9WH564H
```

Lorsqu'on a un message codé et que l'on désire le décoder pour le lire en langage clair, il suffit de procéder de la même manière, en plaçant le message codé dans la chaîne de caractères A$ de la ligne 10 de ce même programme :

```
   1 REM U ?CHR$ H PAUSE ▓RNDU ?
LEN >M ▀?TAN  U ?CHR$ >M ▀?TAN ...
...
  10 LET A$="T6516C9H8CW3HWABHD6
B9WH564H"
  20 PRINT A$
  30 FOR N=1 TO LEN A$
  40 POKE 17152,CODE A$(N)
  50 RAND USR 16514
  60 LET A=PEEK 17154
  70 PRINT CHR$ A;
  80 NEXT N
```

Décodage de ce message

Lorsque ce programme est lancé, l'affichage qui apparaît sur l'écran redonne bien le message original !

```
T6516C9H8CW3HWABHD6B9WH564H
BONJOUR.QUEL.EST.VOTRE.NOM.
```

Avec le micro-ordinateur ZX 81, il est possible de réaliser des programmes codeurs de messages secrets beaucoup plus complexes que celui que nous venons de voir en séparant les fonctions codeur et décodeur.

37 ESCALIER (1 K)

Avec la version de base du micro-ordinateur ZX 81, c'est-à-dire 1 K de mémoire RAM, il est difficile d'afficher directement un dessin sur l'écran à l'aide d'un programme écrit en langage machine, car, dans ce cas, le fichier d'affichage n'existe pas, il se compose simplement de 25 caractères New Line.

Lorsque le module d'extension mémoire est en place, un fichier d'affichage est automatiquement mis en service. Il comporte 792 octets de mémoire RAM, soit 24 lignes de 32 caractères, plus 24 caractères New Line.

Si, à l'aide du programme machine, le code d'un caractère est placé dans un octet du fichier d'affichage, ce caractère apparaît aussitôt sur l'écran.

Le programme qui suit étant prévu pour fonctionner avec 1 K de mémoire RAM, le lancement du programme machine doit être précédé par l'ouverture d'un fichier d'affichage réduit, ce qui sera fait en affichant avec le langage Basic une colonne verticale d'astérisques. Sans cette précaution le micro-ordinateur ZX 81 se bloquerait lorsque le programme machine tenterait d'afficher le premier caractère.

Le programme machine qui suit va afficher un escalier sur l'écran. Il sera facile de modifier ce programme pour obtenir un affichage différent.

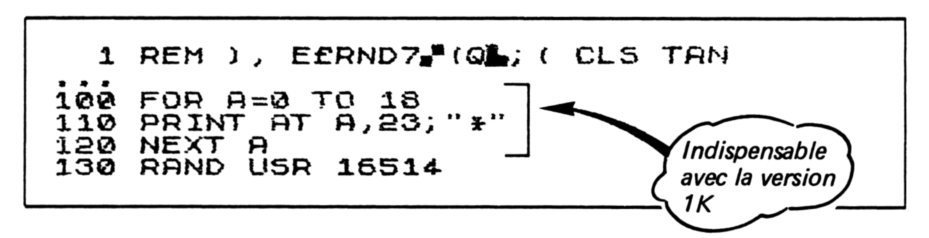
Hexadécimal	**Assembleur**	
11,1A,00	LD DE,00 1A	*Chargement du pas*
2A,0C,40	LD HL, 40 0C	*Chargement de l'adresse de fichier*
23	INC HL	*Incrémentation de HL*
06,10	LD B,10	*Nombre de marches dans B*
→36,82	LD HL ,82	*Dessin de la marche*
19	ADD HL,DE	*Addition de HL et DE*
10,FB	DJNZ,FB	*Décrément de B, si B n'est pas à zéro saut à – 5*
C9	RET	*Retour au Basic*

Le programme Basic chargeur :

```
   1 REM ..................
  10 LET A=16514
  20 LET A$="111A002A0C402306103
6821910FBC9"
  30 FOR B=1 TO LEN A$-1 STEP 2
```

```
40 LET C=CODE A$(B)-28
50 LET D=CODE A$(B+1)-28
60 POKE A,16*C+D
70 LET A=A+1
80 NEXT B
```

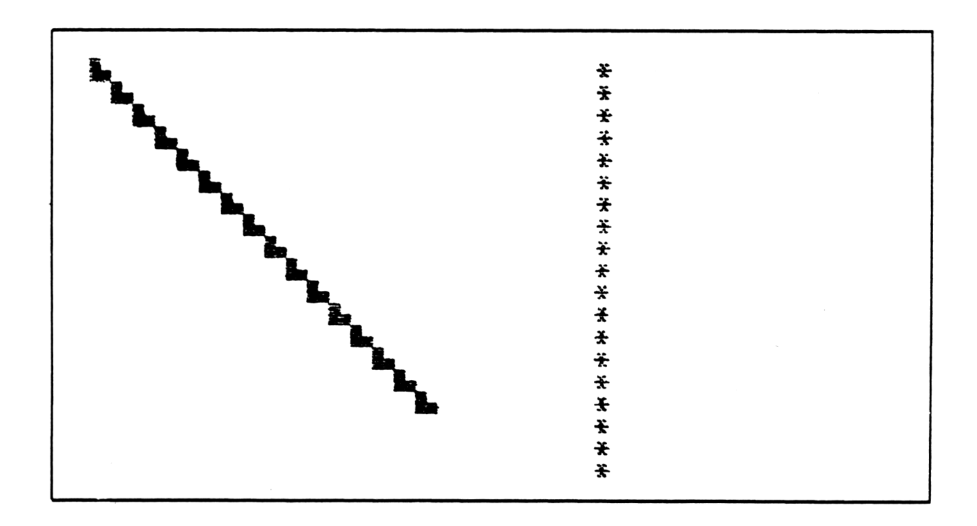
Lorsque ce programme est lancé, le programme machine vient remplacer les points de l'instruction REM de la ligne 1, et il est possible de supprimer les autres lignes du programme devenues inutiles. On peut alors établir le programme définitif tel qu'il est montré dans l'exemple suivant :



```
  1 REM ), E£RND7▪"(Q▙; ( CLS TAN
...
100 FOR A=0 TO 18
110 PRINT AT A,23;"*"
120 NEXT A
130 RAND USR 16514
```


Dans ce programme les lignes 100 à 120 servent à l'établissement de la colonne d'astérisques qui permet l'ouverture d'un fichier d'affichage dans lequel le programme machine va placer le dessin de l'escalier.

Exemple d'utilisation :



38 TEST DU CLAVIER (1 K)

Il existe dans la mémoire ROM du micro-ordinateur ZX 81 un sous-programme qui permet de tester si une touche du clavier a été pressée.

Ce sous-programme, qui se trouve à l'adresse 699 ou 02 BB en hexadécimal, place dans la paire de registres HL un nombre correspondant à la touche pressée.

Nous allons établir un programme machine qui va tester si une touche du clavier est pressée. Dans le cas où une touche est pressée ce programme machine provoque l'affichage sur l'écran du contenu de la paire de registre HL qui correspond à la touche pressée.

L'intérêt de ce programme sera de nous familiariser avec le contrôle des touches du clavier en langage machine. Nous allons voir par la suite des programmes machine sur le même principe qui nous permettront de contrôler des mouvements rapides sur l'écran à l'aide des touches du clavier.

Voici ce programme machine :

Hexadécimal	Assembleur	
CD,BB,02	CALL 02,BB	*Test clavier*
44	LD B,H	*Transfert de H dans B*
4D	LD C,L	*Transfert de L dans C*
7C	LD A,H	*Transfert de H dans A*
A5	AND L	*Opération logique AND entre A et L*
3C	INC A	*Incrémentation de A*
28,F6	JR Z,F6	*Saut à − 10 si le résultat est 0*
C9	LET	*Retour au Basic*

Lorsque aucune touche n'est pressée le contenu de HL est FF FF.

L'opération logique AND place alors l'octet FF dans le registre A. Dans ce cas après incrémentation le contenu de A passe à zéro ce qui permet de reboucler le programme machine à son début si aucune touche n'est pressée.

Le programme Basic chargeur :

```
   1 REM ............
  10 LET A=16514
  20 LET A$="CDBB02444D7CA53C28F
6C9"
  30 FOR B=1 TO LEN A$-1 STEP 2
```

```
40 LET C=CODE A$(B)-28
50 LET D=CODE A$(B+1)-28
60 POKE A,16*C+D
70 LET A=A+1
80 NEXT B
```

Le lancement de ce programme chargeur place le programme en langage machine dans l'instruction REM de la ligne 1, ce qui permet de supprimer les autres lignes du programme chargeur devenues inutiles. Nous ajouterons une instruction qui affichera le contenu des registres H et L, et une instruction de saut créant une boucle :

```
 1 REM LN ▩▀???▤WC PLOT TAN .
10 PRINT USR 16514
20 GOTO 10
```

Lorsque ce programme est lancé et qu'une touche est pressée, un nombre correspondant à cette touche vient s'afficher sur l'écran.

En laissant le doigt pressé sur une touche le nombre correspondant à cette touche s'affiche à répétition le programme s'arrête lorsque chacune des 22 lignes de l'écran contient un nombre correspondant à une touche pressée.

L'exemple suivant montre l'affichage sur l'écran lorsque le programme est lancé et que l'on a pressé les touches A, B, C, D etc... dans l'ordre alphabétique ;

```
64959
65021
57215
61438
63485
63483
61437
57341
57279
63455
61375
63423
64447
63359
61311
64479
64991
65019
61435
64509
57339
61407
```

39 AFFICHAGE COMMANDÉ PAR LE CLAVIER (1 K)

Le programme suivant sera une application de la scrutation du clavier à l'aide du langage machine.

Une fois lancé, ce programme affichera sur l'écran une demi-ligne d'étoiles en vidéo inversée, chaque fois que la touche 8 sera pressée, la pression sur la touche BREAK arrêtera le programme, la pression de toute autre touche du clavier restera sans effet.

Voici le programme en langage machine :

Hexadécimal	Assembleur	
→D5	PUSH DE	*Mise en mémoire de DE dans la pile*
CD,BB,02	CALL 02,BB	*Appel du Test du clavier*
7D	LD A,L	*Transfert de L dans A*
3C	INC A	*Incrémentation de A*
D1	POP DE	*Retour de DE à sa valeur*
—28,F7	JR Z,F7	*Si l'indication est à 0 saut à −9*
D5	PUSH DE	*Mise de DE dans la pile*
E5	PUSH HL	*Mise de HL dans la pile*
C1	POP BC	*La valeur de HL passe dans BC*
CD,BD,07	CALL 07,BD	*Appel du code du caractère*
D1	POP DE	*Retour de DE à sa valeur*
7E	LD A, HL	*Chargement de A*
2A,1E,40	LD HL, 40 1E	*Adresse du RAM TOP dans HL*
FE,24	CP,24	*Comparaison du caractère 8*
—28,01	JR Z,01	*Si le résultat est zéro saut à +1*
C9	RET	*Retour au Basic*
→06,10	LD B,10	*Chargement de 16 dans B*
3E,97	LD A,97	*Chargement du caractère affiché dans A*
→00	NOP	*Pas d'opération*
D7	RST	*Affichage du caractère*
—10,FC	DJNZ,FC	*Décrémentation de B et saut à −4 si pas 0*
C9	RET	*Retour au Basic*

Ce programme machine fait appel à deux sous-programmes qui sont placés dans la mémoire ROM du micro-ordinateur ZX 81. A l'adresse 699 soit 02 BB en hexadécimal nous utilisons le sous-programme de test des touches du clavier. A l'adresse 1981 soit 07 BD le sous-programme de recherche du code du caractère dont la touche est pressée.

Ces sous-programmes modifient parfois le contenu de la paire de registres DE. Pour être certain de conserver le contenu de ces registres, celui-ci est placé dans la pile avant l'appel du sous-programme pour être replacé dans la paire DE après l'appel du sous-programme.

Le programme Basic chargeur :

```
   1 REM ........................
......................
  10 LET A=16514
  20 LET A$="D5CDBB027D3CD128F7D
5E5C1CDBD07D17E2A0E40FE242801C90
6103E97 00D710FCC9"
  30 FOR B=1 TO LEN A$-1 STEP 2
  40 LET C=CODE A$(B)-28
  50 LET D=CODE A$(B+1)-28
  60 POKE A,16*C+D
  70 LET A=A+1
  80 NEXT B
```

Lorsque ce programme chargeur est lancé, le programme en langage machine passe dans l'instruction REM de la ligne 1 en remplacement des points.

L'exemple suivant montre le programme définitif complété à la suite de la ligne 1. Les lignes 20 et 30 de ce programme assurent une temporisation qui est indispensable car, étant donné la rapidité d'exécution des programmes ; en langage machine, chaque pression sur la touche 8, même très brève, provoquerait l'affichage de plusieurs demi-lignes de caractères.

```
   1 REM STR$ LN ■▀?WSGN C RUN S
TR$ FAST AT LN ■▀SGN C▀TAN ▞(Y■
NOT ( UNPLOT TAN .........
  10 RAND USR 16514
  20 FOR A=1 TO 20
  30 NEXT A
  40 GOTO 10
```

Lorsque ce programme est lancé, chaque pression sur la touche 8 du clavier provoque l'affichage sur l'écran d'une demi-ligne d'étoiles inversées.

L'exemple qui suit montre l'affichage sur l'écran après 7 pressions sur la touche 8.

```
********************************
********************************
********************************
****************
```

LES CARACTÈRES GÉANTS

Dans la mémoire ROM du micro-ordinateur ZX 81 on trouve le code des divers caractères utilisés par le ZX 81 pour l'affichage sur l'écran.

Ce jeu de caractères est recopié dans la mémoire à partir de l'adresse 7680 ou 1E 00 en hexadécimal. Chaque caractère est défini par 8 octets successifs. Dans chaque octet les espaces noirs correspondent à des bits à 1 et les espaces blancs à des bits à 0. Le caractère A par exemple correspond aux 8 octets occupant les adresses 7984 à 7991 (1E 30 à 1E 37 en hexadécimal).

En temps normal ces octets sont utilisés pour afficher 24 lignes de 32 caractères. Il est possible de modifier les octets constituant le jeu de caractères pour afficher, par exemple, des caractères semblables mais plus grands.

Il est ainsi possible d'afficher 6 lignes de 8 caractères en multipliant la taille des caractères par 4 ou 3 lignes de caractères plus grands ne comportant plus que 4 caractères par ligne, ceci à condition d'utiliser l'extension mémoire.

Comme nous nous efforçons dans nos programmes en langage machine d'utiliser la version de base du ZX 81 qui ne comporte que 1 K de mémoire RAM, nous nous contenterons d'afficher un seul caractère géant.

Nous allons établir un même programme affichant le caractère géant dont la touche est pressée, ce programme sera donné en deux langages, en langage machine et en langage Basic.

L'intérêt de donner ces deux programmes sera de nous permettre de comparer la vitesse d'exécution d'un programme semblable écrit en langage évolué et en langage machine.

40 PROGRAMME BASIC : CARACTERE GEANT (1 K)

Lorsque ce programme est lancé l'écran reste vide en attendant qu'une touche du clavier soit pressée.

Quand une touche est pressée, le caractère géant qui correspond à cette touche vient s'afficher sur l'écran, il y reste affiché pendant environ deux secondes avant de s'effacer. Il faut ensuite presser une nouvelle fois sur une autre touche du clavier pour faire apparaître un autre caractère géant.

```
 10 IF INKEY$<>"" THEN GOTO 10
 20 IF INKEY$="" THEN GOTO 20
 40 LET A=CODE INKEY$
 50 GOSUB 140
 60 FOR A=0 TO 100
 70 NEXT A
 75 CLS
 80 GOTO 20
140 FOR B=2 TO 9
145 LET R=PEEK (7680+8*A+B-2)
150 LET S=128
155 FOR C=8 TO 15
160 LET Z=INT (R/S)
165 LET R=R-Z*S
170 LET S=S/2
175 IF Z THEN PRINT AT B,C;"█"
180 NEXT C
185 NEXT B
190 RETURN
```

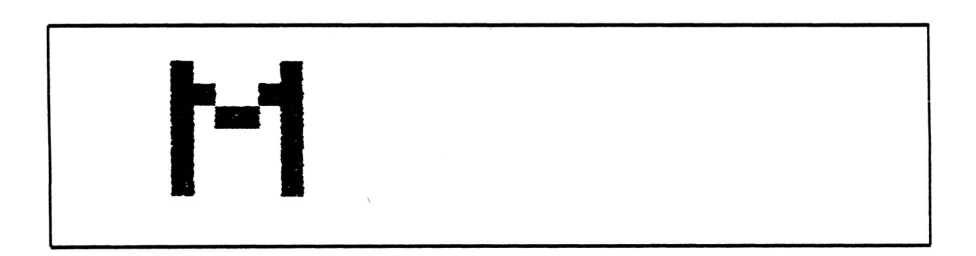
Boucle attendant que l'on ait relaché la touche NEWLINE au lancement

Attente d'une frappe de touche

Lecture des informations concernant une ligne du caractère

Décomposition en 8 bits

L'exemple qui suit montre l'affichage sur l'écran lorsque la touche correspondant à la lettre M vient d'être pressée.





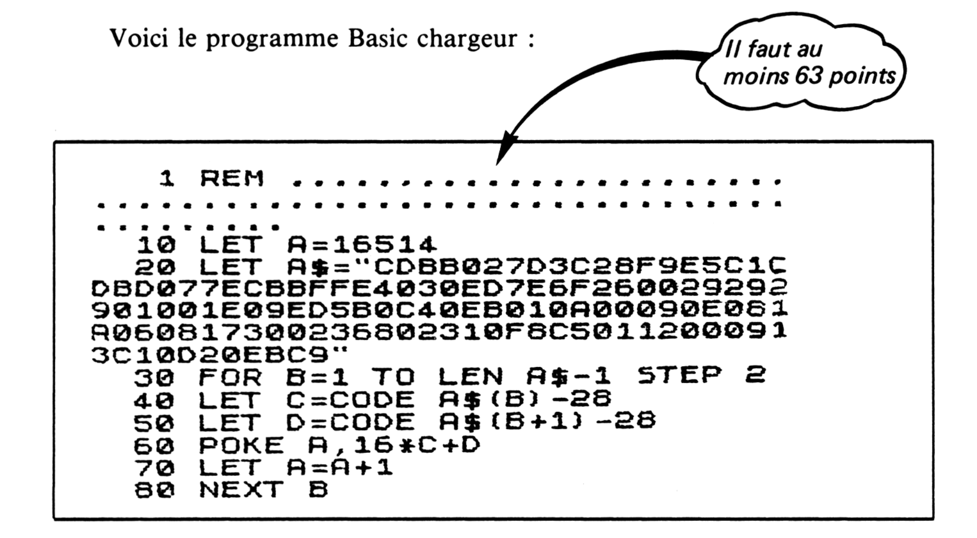
PROGRAMME EN LANGAGE MACHINE : CARACTERE GEANT (1 K)

Ce programme en langage machine va donner le même affichage sur l'écran lorsque la même touche du clavier sera pressée. Comme le précédent, il débute par le test du clavier pour vérifier si une touche est pressée.

Hexadécimal	Assembleur	
CD,BB,02	CALL 02,BB	*Test du clavier*
7D	LD A,L	*Charger L dans A*
3C	INC A	*Incrémenter A*
28,F9	JR Z,F9	*Saut à – 7 si zéro*
E5	PUSH HL	*HL dans la pile*
C1	POP BC	*HL dans BC*
CD,BD,07	CALL 07,BD	*Code des caractères*
7E	LD A, HL	*Charger dans A l'octet pointé par HL*
CB,BF	RES 7,A	*Mise à zéro du bit 7 de A*
FE,40	CP 40	*Comparaison avec 40*
30,ED	JR NC,ED	*Saut à – 19 si pas de retenue*
7E	LD A, HL	*Charger dans A l'octet pointé par HL*
6F	LD L,A	*Charger A dans L*
26,00	LD H,00	*Charger 00 dans H*
29	ADD HL,HL	*Multiplier HL par 8*
29	ADD HL,HL	
29	ADD HL,HL	
01,00,1E	LD BC,1E 00	*Adresse du début des caractères dans BC*
09	ADD HL,BC	*Addition de HL et BC*
ED,5B,0C,40	LD DE, 40 0C	*L'octet à 40 0C dans E*
EB	EX DE,HL	*Echange des registres HL et DE*
01,0A,00	LD BC,00 0A	*Charger 00 0A dans BC*
09	ADD HL,BC	*Addition de HL et BC*
0E,08	LD C,08	*Charger 08 dans C*
1A	LD A, DE	*Charger dans A l'octet pointé par DE*
06,08	LD B,08	*Charger 08 dans B*
17	RLA	*Rotation à gauche de l'octet dans A*
30,02	JR NC,02	*Si pas de retenue saut à + 2*
36,80	LD HL ,80	*Charger 80 à l'adresse pointée par HL*
23	INC HL)	*Incrémenter HL*
10,F8	DJNZ F8	*Décrémenter B si non zéro saut à – 8*

C5	PUSH BC	*Placer BC dans la pile*
01,12,00	LD BC,00 12	*Charger 0012 dans BC*
09	ADD HL,BC	*Addition de HL et BC*
13	INC DE	*Incrémentation de DE*
C1	POP BC	*Remet BC à sa valeur primitive*
0D	DEC C	*Décrémente C*
20,EB	JR NZ,EB	*Si non à zéro saut à −21*
C9	RET	*Retour au Basic*

Voici le programme Basic chargeur :




```
   1 REM ........................
....................................
.........
  10 LET A=16514
  20 LET A$="CDBB027D3C28F9E5C1C
DBD077ECBBFFE4030ED7E6F260029292
901001E09ED5B0C40EB010A00090E061
A060817300236802310F8C5011200091
3C10D20EBC9"
  30 FOR B=1 TO LEN A$-1 STEP 2
  40 LET C=CODE A$(B)-28
  50 LET D=CODE A$(B+1)-28
  60 POKE A,16*C+D
  70 LET A=A+1
  80 NEXT B
```

Lorsque ce programme chargeur est lancé par une instruction RUN, le programme machine vient remplacer 63 points dans l'instruction REM de la ligne 1. On peut alors supprimer les lignes 10 à 80 et écrire le programme définitif à la suite de la ligne 1.

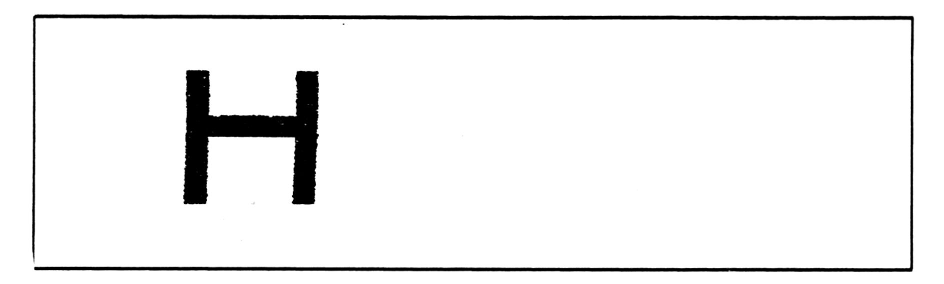
```
   1 REM LN ▇▘?WC RAND FAST AT L
N▝▛ GOSUB D▘ 2▒ GOSUB ?£RND FOR
▘▒ ▒:▒,▟▒*K ▘Q▇7( SAVE VAL ▘> ▒<
AT $4 FOR TAN .
  10 FOR A=0 TO 10
  20 PRINT AT A,24;" "
  30 NEXT A
  40 RAND USR 16514
  50 FOR A=0 TO 100
  60 NEXT A
  70 CLS
  80 GOTO 10
```

Ouverture du fichier d'affichage, nécessaire avec 1K

Comme avec le programme en langage Basic lorsque le programme précédent en langage machine est lancé, l'écran reste vide en attendant qu'une touche du clavier soit pressée.

Lorsqu'une touche est pressée, le caractère géant correspondant à cette touche vient s'afficher sur l'écran pendant environ deux secondes.

L'exemple qui suit montre l'affichage sur l'écran lorsque la touche H vient d'être pressée :



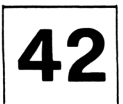
Bien que les deux programmes en langage Basic et en langage machine que nous venons de voir donnent le même résultat vous avez sûrement remarqué que le programme machine est beaucoup plus rapide dans son exécution, l'affichage du caractère se faisant pratiquement d'une manière instantanée, alors que son équivalent Basic prend plusieurs secondes.

INVERSION VIDÉO

Normalement l'affichage du micro-ordinateur ZX 81 se fait en lettres noires sur fond blanc.

Dans le commerce on trouve des modules qui peuvent être montés dans le boîtier du ZX 81 afin d'obtenir des lettres blanches sur fond noir.

Il est plus élégant cependant d'opérer cette inversion vidéo à l'aide d'un programme en langage machine.



INVERSION VIDÉO SUR 6 LIGNES (1 K)

Le programme machine qui suit met en vidéo inversée l'affichage des 6 premières lignes de l'écran. Ce programme est prévu pour fonctionner avec la version 1 K du micro-ordinateur ZX 81.

Hexadécimal	Assembleur	
2A,0C,40	LD HL, 40 0C	*Chargement de 16384 dans HL*
06,C0	LD B,C0	*Chargement de 192 dans B*
00	NOP	*Pas d'opération*
3E,76	LD A,76	*Caractère NEW LINE dans A*
23	INC HL	*Pointage du caractère suivant*
BE	CP HL	*Comparaison avec NEW LINE*
28,FC	JR Z,FC	*Si zéro saut à – 4*
3E,80	LD A,80	*Inversion dans A*
AE	XOR HL ,A	*Inversion du caractère pointé*
77	LD HL ,A	*Affichage du caractère inversé*
10,F3	DJNZ F3	*Décrément de B, si non zéro saut – 13*
C9	RET	*Retour au Basic*

Programme Basic chargeur :

```
   1 REM .........................
  10 LET A=16514
  20 LET A$="2A0C4006C0003E7623E
E28FC3E80AE7710F3C9"
  30 FOR B=1 TO LEN A$-1 STEP 2
  40 LET C=CODE A$(B)-28
  50 LET D=CODE A$(B+1)-28
  60 POKE A,16*C+D
  70 LET A=A+1
  80 NEXT B
```

Lorsque ce programme est lancé, le programme machine vient remplacer les points dans l'instruction REM de la ligne 1, que l'on peut placer en tête d'un programme quelconque, et chaque fois que ce programme mettra en service le programme machine à l'aide d'une instruction contenant USR 16514, l'affichage des 6 premières lignes de l'écran apparaîtra en vidéo inversée.

Le programme suivant montre une application du programme machine.

```
   1 REM E£RND▪"" Y
9150 Y▮?( NEXT TAN ....
  10 INPUT A$
  20 PRINT A$
  30 RAND USR 16514
```

Le résultat sur l'écran est plus élégant que la recopie de l'imprimante :

```
---- MICRO ORDINATEUR ----

      ... ZX 81 ...
```

43 INVERSION VIDÉO DE L'ÉCRAN (16 K)

Si l'on dispose d'un module d'extension mémoire, on peut établir un programme machine semblable qui inverse la surface entière de l'écran.

Voici le programme machine d'inversion vidéo pour ZX 81 avec 16 K mémoire :

Hexadécimal	**Assembleur**	
2A,0C,40	LD HL, 40 0C	*Charge l'adresse de début du fichier d'affichage dans HL*
0E,16	LD C,16	*Nombre de lignes dans C*
→ 06,20	LD B,20	*Nombre de caractères par ligne dans B*
→ 23	INC,HL	*Incrémente HL*
7E	LD A, HL	*Charger A avec le caractère*
C6,80	ADD A,80	*Inversion du caractère*
77	LD HL ,A	*Afficher le caractère inversé*
10,F9	DJNZ F9	*Décrémente B saut à – 7 si non zéro*
23	INC HL	*Incrémente HL (saut de NEW LINE)*
0D	DEC,C	*Ligne suivante*
20,F3	JR NZ,F3	*Si C n'est pas à zéro saut à – 13*
C9	RET	*Retour au Basic*

Programme Basic chargeur :

```
   1 REM ........................
  10 LET A=16514
  20 LET A$="2A0C400E160620237EC
6807710F9230D20F3C9"
  30 FOR B=1 TO LEN A$-1 STEP 2
  40 LET C=CODE A$(B)-28
  50 LET D=CODE A$(B+1)-28
  60 POKE A,16*C+D
  70 LET A=A+1
  80 NEXT B
```

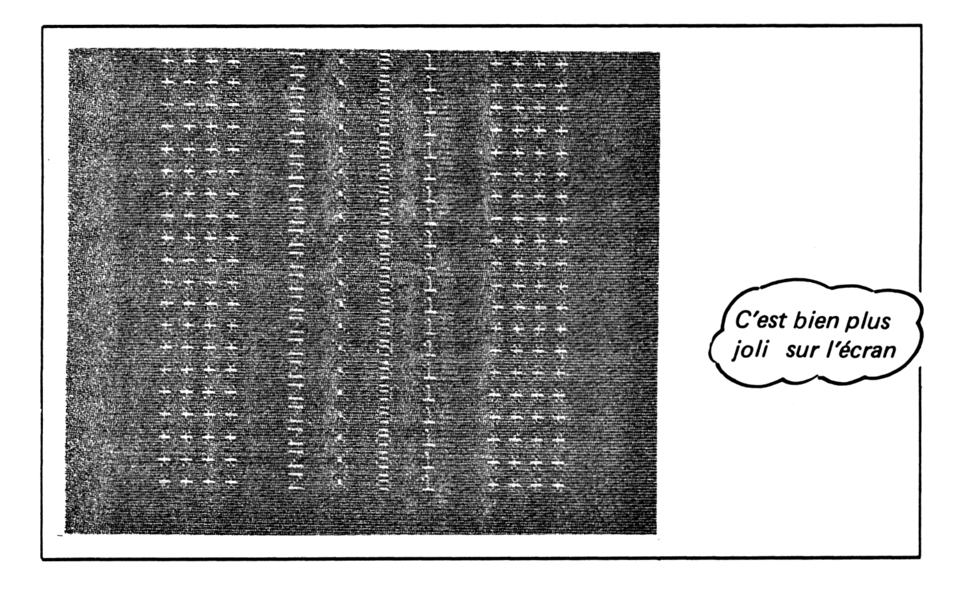
Comme avec le programme machine précédent, la ligne 1 du programme chargeur qui contient le programme en langage machine peut être placé en tête d'un programme afin de permettre à tout moment d'inverser l'affichage vidéo sur l'écran.

Le petit programme qui suit montre une des applications possibles de ce programme machine.

En réponse vous taperez ce qui vous passe par la tête, si ça tient sur une ligne.

```
   1 REM E£RND:-▪477$4 NEXT TAN
.....
  10 INPUT A$
  20 FOR A=1 TO 20
  30 PRINT A$
  40 NEXT A
  50 RAND USR 16514
```

On voit, dans l'exemple ci-dessous, un affichage en vidéo inversée qui résulte de l'utilisation de ce programme.



Etudiez les différences entre les deux programmes d'inversion vidéo, elles vous aideront à mieux comprendre la programmation en langage machine.



DOUBLE COMMANDE PAR LE CLAVIER (1 K)

Le programme machine qui suit utilise le test des touches du clavier en langage machine que nous avons déjà vu. Dans ce programme nous utiliserons deux touches pour réaliser deux commandes différentes.

Nous avons choisi les touches 5 et 8 car celles-ci sont très souvent utilisées pour programmer des déplacements dans le sens des flèches qui figurent sur ces touches. Mais nous pouvons programmer d'autres touches en utilisant le code hexadécimal de ces touches (annexe A du manuel Sinclair de programmation du ZX 81).

Lorsque le programme en langage machine qui suit est en service, une pression sur la touche 5 affiche une ligne d'astérisques et une pression sur la touche 8 affiche une ligne noire sur l'écran.

Voici ce programme machine :

Hexadécimal	Assembleur	
D5	PUSH DE	*DE dans la pile*
CD,BB,02	CALL 02,BB	*Appel du test clavier*
7D	LD A,L	*L dans A*
3C	INC A	*Incrémente A*
D1	POP DE	*Retour de DE*
28,F7	JRZ,F7	*Si zéro saut à −9*
D5	PUSH DE	*DE dans la pile*
E5	PUSH HL	*HL dans la pile*
C1	POP BC	*Valeur de HL dans BC*
CD,BD,07	CALL 07,BD	*Code des caractères*
D1	POP DE	*Retour de DE*
7E	LD A, HL	*Chargement du caractère dans A*
2A,0E,40	LD HL, 40 0E	*Contenu de l'adresse 16398 dans HL*
FE,21	CP 21	*Comparaison de H avec le caractère 5*
28,0D	JR Z,0D	*Si zéro saut à +13*
FE,24	CP 24	*Comparaison avec le caractère 8*
28,01	JR Z,01	*Saut de +1 si zéro*
C9	RET	*Retour au Basic*
06,20	LD B,20	*Nombre de caractères d'une ligne*

3E,83	LD A,83	*Caractère du trait noir*
D7	RST	*Affichage du caractère*
10,FD	DJNZ FD	*Décrément B et saut à –3 si non 0*
C9	RET	*Retour au Basic*
00,00	NOP,NOP	*Pas d'opération*
06,20	LD B,20	*Nombre de caractère d'une ligne*
3E,17	LD A,17	*Caractère de l'astérisque*
00	NOP	*Pas d'opération*
D7	RST	*Affichage du caractère*
10,FD	DJNZ,FD	*Décrément B saut à –3 si non zéro*
C9	RET	*Retour au Basic*

Programme Basic chargeur :

```
   1 REM ........................
..............................
  10 LET A=16514
  20 LET A$="D5CDBB027D3CD128F7D
5E5C1CD8D07D17E2A0E40FE21280DFE2
42801C906203E83D710FDC9000006203
E1700D710FDC9"
  30 FOR B=1 TO LEN A$-1 STEP 2
  40 LET C=CODE A$(B)-28
  50 LET D=CODE A$(B+1)-28
  60 POKE A,16*C+D
  70 LET A=A+1
  80 NEXT B
```

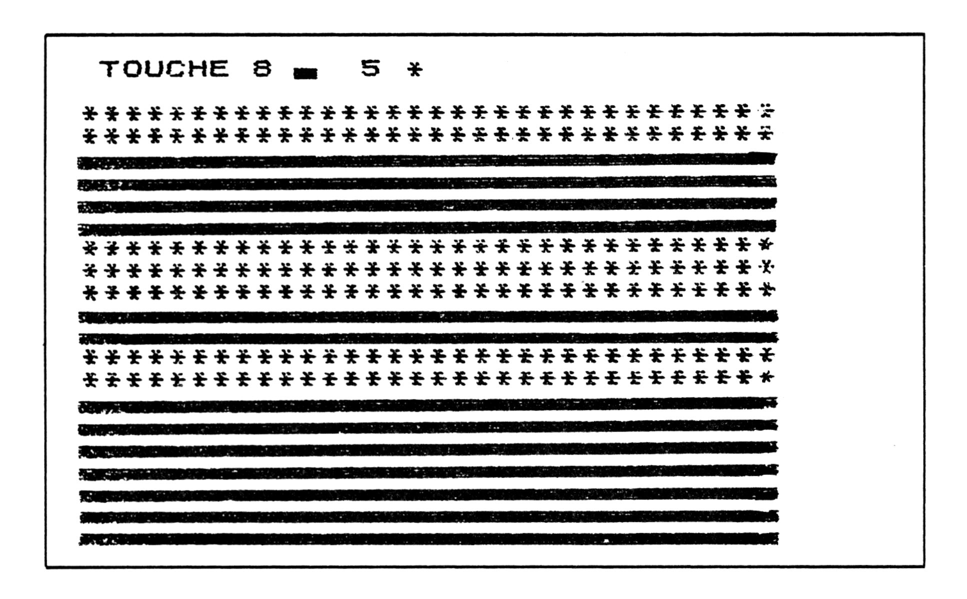
Lorsque ce programme chargeur est lancé, le programme en langage machine vient remplacer les points de l'instruction REM de la ligne 1 du programme chargeur. Les autres lignes du programme chargeur qui sont devenues inutiles sont alors supprimées.

La ligne 1 restant qui contient le programme en langage machine est alors placée en tête du petit programme d'application qui suit.

```
   1 REM STR$ LN ▀?WSGN C RUN S
TR$ FAST AT LN ▀SGN C$ RETURN 8
C TAN 4Y NOT ( CLEAR TAN  4Y*
 NOT ( CLEAR TAN .......
  10 PRINT " TOUCHE 8 ▄  5 *"
  15 PRINT
  20 RAND USR 16514
  30 GOTO 20
```

Lorsque ce programme est lancé, on voit s'inscrire sur l'écran le numéro des deux touches actives ainsi que les caractères qui sont affichés par ces deux touches.

L'exemple qui suit montre l'affichage qui résulte de pressions successives sur les touches 5 et 8 du clavier.



45 DESSIN D'UN RECTANGLE (1 K)

Le programme machine qui suit va dessiner un rectangle sur l'écran. Pour ce programme nous utiliserons la routine RST placée en mémoire ROM ; ceci nous permettra de tracer le dessin sans nous préoccuper de la taille du fichier d'affichage si nous ne disposons que de 1 K de mémoire.

Voici le programme machine :

Hexadécimal	Assembleur	
06,08	LD B,08	*Nombre d'espaces blancs*
3E,00	LD A,00	*Caractère d'espacement dans A*
D7	RST	*Affichage du caractère*
10,FD	DJNZ FD	*Décrément B saut à −3 si non zéro*
06,10	LD B,10	*Nombre de caractères horizontaux*
3E,83	LD A,83	*1er caractère du tracé horizontal dans A*
D7	RST	*Affichage du caractère*
10,FD	DJNZ FD	*Décrément B saut à −3 si non zéro*
0E,08	LD C,08	*Nombre de caractères verticaux*
00	NOP	*Pas d'opération*
06,10	LD B,10	*Nombre d'espaces en blanc*
3E,00	LD A,00	*Caractère d'espacement dans A*
D7	RST	*Affichage du caractère*
10,FD	DJNZ FD	*Décrément de B saut à −3 si non zéro*
3E,05	LD A,05	*1er caractère vertical dans A*
D7	RST	*Affichage du caractère*
06,0E	LD B,0E	*Nombre d'espaces blancs*
3E,00	LD A,00	*Caractère d'espacement dans A*
D7	RST	*Affichage du caractère*
10,FD	DJNZ FD	*Décrément de B saut à −3 si non zéro*
3E,85	LD A,85	*2e caractère vertical dans A*
D7	RST	*Affichage du caractère*
0D	DEC C	*Ligne verticale suivante*
20,E9	JR NZ,E9	*Saut à −23 si non zéro*
06,10	LD B,10	*Nombre d'espaces blancs*
3E,00	LD A,00	*Caractère d'espacement dans A*
D7	RST	*Impression du caractère*
10,FD	DJNZ FD	*Décrément et saut à −3 si non zéro*
06,10	LD B,10	*Nombre de caractères horizontaux*
3E,03	LD A,03	*2e caractère horizontal dans A*
D7	RST	*Impression du caractère*
10,FD	DJNZ FD	*Décrément et saut à −3 si non zéro*
C9	RET	*Retour au Basic*

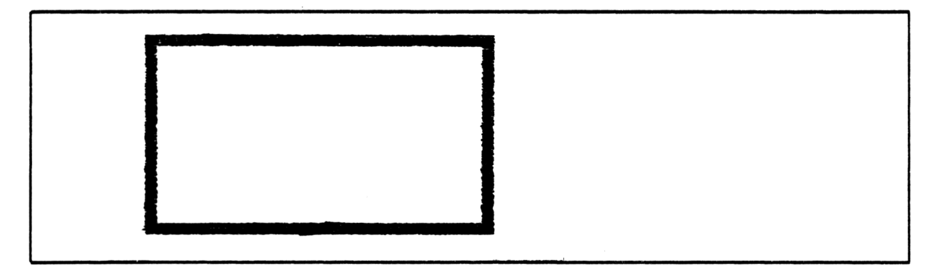
Programme Basic chargeur :

```
   1 REM ..........................
.................................
.
  10 LET A=16514
  20 LET A$="06083E00D710FD06103
E83D710FD0E0800061O3E00D710FD3E0
5D7060E3E00D710FD3E85D70D20E9061
03E00D710FD06103E03D710FDC9"
  30 FOR B=1 TO LEN A$-1 STEP 2
  40 LET C=CODE A$(B)-28
  50 LET D=CODE A$(B+1)-28
  60 POKE A,16*C+D
  70 LET A=A+1
  80 NEXT B
```

Programme utilisateur :

```
   1 REM ▞▒Y NOT ( CLEAR ▞(Y▄NOT
 ( CLEAR :▒ ▞(Y NOT ( CLEAR Y▌NO
T ▞:Y NOT ( CLEAR Y▌NOT $4 DIM ▞
(Y NOT ( CLEAR ▞(Y▀NOT ( CLEAR T
AN .
  10 RAND USR 16514
```

Lorsque ce programme est lancé, le rectangle dessiné par le programme en langage machine vient s'afficher sur l'écran :



Bien qu'il soit possible d'afficher toutes sortes de dessins sur l'écran du téléviseur à l'aide du langage machine. Il est souvent plus pratique d'utiliser le langage Basic pour afficher des dessins statiques, le langage machine étant utilisé pour les dessins en mouvement où la rapidité des mouvements permise par le langage machine trouve son application.

Le programme suivant en est une illustration.



46 PING PONG (1 K)

Le programme machine qui suit est intéressant car il montre l'utilisation d'un sous-programme en langage machine dans un programme principal en langage Basic.

Le programme qui suit montre les mouvements de va et vient d'une étoile dans un cadre dessiné sur l'écran.

Voici le sous-programme en langage machine.

Hexadécimal	Assembleur	
2A,0C,40	LD HL, 40 0C	*Début du ficher d'affichage dans HL*
01,83,00	LD BC,00 83	*Emplacement de l'étoile*
09	ADD HL,BC	*Addition de HL et BC*
36,17	LD HL, 17	*Affichage de l'étoile*
16,20	LD D,20	*1re temporisation*
1E,80	LD E,80	*2e temporisation*
1D	DEC E	*Décrément 2e temporisation*
20,FD	JR NZ,FD	*Saut à −3 si non zéro*
15	DEC D	*Décrément 1re temporisation*
20,F8	JR NZ,F8	*Saut à −8 si non zéro*
36,00	LD HL, 00	*Effacement de l'étoile*
23	INC HL	*Avance d'un pas*
7E	LD A,HL	*Caractère indiqué par HL dans A*
B7	OR A	*OU logique de A*
C0	RET NZ	*Retour au Basic si A non nul*
18,EC	JR EC	*Saut à −20*

Dans ce sous-programme le déplacement de l'étoile n'est pas très rapide, mais il est facile d'accélérer ce mouvement en diminuant la valeur du nombre chargé dans le registre D pour la première temporisation par exemple.

Programme Basic chargeur :

```
   1 REM .......................
........
  10 LET A=16514
  20 LET A$="2A0C4001830009361 71
6201E801D20FD1520F83600237EB7C01
8EC"
  30 FOR B=1 TO LEN A$-1 STEP 2
  40 LET C=CODE A$(B)-28
  50 LET D=CODE A$(B+1)-28
  60 POKE A,16*C+D
  70 LET A=A+1
  80 NEXT B
```

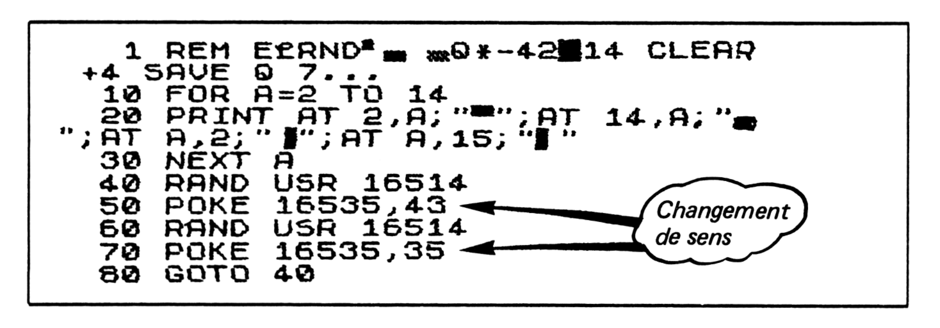
Lorsque ce programme chargeur est lancé, le programme en langage machine vient remplacer les points de l'instruction REM de la ligne 1 du programme chargeur.

Cette ligne est alors conservée pour servir de sous-programme au programme Basic, les autres lignes devenues inutiles du programme chargeur sont alors supprimées.

Ce programme Basic commence par tracer le cadre où évoluera l'étoile, avant de mettre en service le sous-programme qui détermine les mouvements de l'étoile. Lorsque l'étoile arrive au bord du cadre, le sous-programme repasse la main au langage Basic, qui inverse le sens du déplacement de l'étoile par une instruction Basic POKE. Celle-ci transforme l'incrémentation de la paire de registres HL en décrémentation et vice-versa, ce qui fait repartir l'étoile dans la direction opposée.

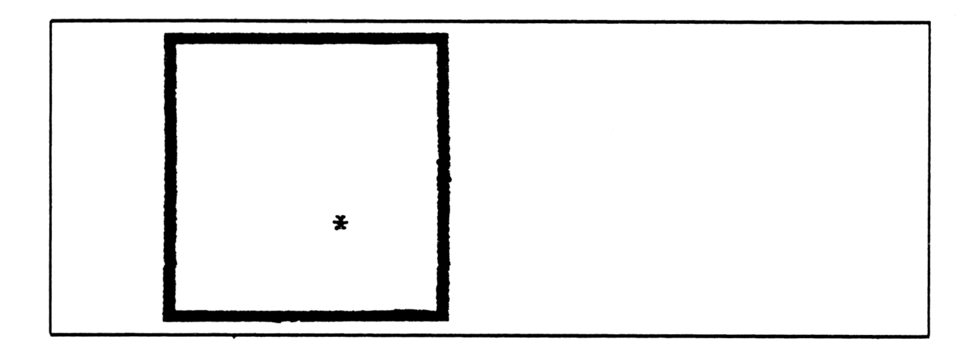
Ce programme a pour but de montrer qu'il est possible d'intervenir sur le texte d'un sous-programme en langage machine en cours d'exécution à partir du programme principal écrit en langage Basic.

Voici le programme Ping Pong.



```
  1 REM E£RND▀▄ ▒Q*-42█14 CLEAR
+4 SAVE Q 7...
 10 FOR A=2 TO 14
 20 PRINT AT 2,A;"▀";AT 14,A;"▄
";AT A,2;"█";AT A,15;"█"
 30 NEXT A
 40 RAND USR 16514
 50 POKE 16535,43
 60 RAND USR 16514
 70 POKE 16535,35
 80 GOTO 40
```


Lorsque ce programme est lancé, le cadre s'affiche sur l'écran et l'étoile parcourt un va et vient d'un bord à l'autre du cadre à la manière d'une balle de ping pong, comme on aimerait vous le montrer ici !



47 LA RAQUETTE (1 K)

Le programme machine qui suit va commander les mouvements d'une raquette qui se déplace entre deux barres situées en haut de l'écran. Une pression sur la touche 8 dirige la raquette vers la droite et une pression sur la touche 5 dirige la raquette vers la gauche dans le sens des flèches gravées sur ces touches.

Ce programme sera utilisé lorsqu'on voudra programmer des jeux en langage machine comme les jeux de tennis ou de squash.

A cause de la programmation en langage machine, les mouvements de la raquette sont très rapides, mais ceux-ci peuvent être rendus encore plus rapides en diminuant les temporisations qui les ralentissent.

Voici ce programme machine constitué de la réunion de plusieurs programmes vus précédemment.

Hexadécimal	Assembleur	
3E,85	LD A,85	*Charger dans A le 1er caractère*
D7	RST	*Affichage du 1er caractère*
06,1E	LD B,15	*Nombre d'espaces*
3E,00	LD A,00	*Caractère nul dans A*
┌→ D7	RST	*Affichage de l'espace*
└─ 10,FD	DJNZ FD	*Si B n'est pas à zéro saut à −3*
3E,05	LD A,05	*Charger dans A le 2e caractère*
D7	RST	*Affichage du 2e caractère*
2A,0E,40	LD HL, 40 0E	*Adresse de l'affichage dans HL*
01,F0,FF	LD BC,FF F0	*Charger −16 dans BC*
09	ADD HL,BC	*Addition de HL et BC*
22,0E,40	LD 40 0E ,HL	*Charger HL à l'adresse 40 0E*
3E,83	LD A,83	*Caractère de la raquette dans A*
D7	RST	*Affichage du caractère*
00	NOP	*Pas d'opération*
┌→ D5	PUSH DE	*DE dans la pile*
CD,BB,02	CALL 02,BB	*Test Clavier*
7D	LD A,L	*Chargement de L dans A*
3C	INC A	*Incrémentation de A*
D1	POP DE	*La tête de pile dans DE*
└─ 28,F7	JR Z,F7	*Saut à −9 si zéro*
D5	PUSH DE	*DE dans la pile*
E5	PUSH HL	*HL dans la pile*
C1	POP BC	*BC reçoit la tête de pile*
CD,BD,07	CALL 07,BD	*Code des caractères*
D1	POP DE	*Tête de la pile dans DE*

```
┌──7E          LD A, HL          Caractère du clavier dans A
   2A,0E,40    LD HL, 40 0E      Charger l'octet situé à 40 0E dans HL
   FE,21       CP 21             Comparer A avec le caractère 5
┌──28,1D       JR Z,1D           Si zéro saut à +29
│  FE,24       CP 24             Comparer H avec le caractère 8
│┌─28,01       JR Z,01           Si zéro saut à +1
││ C9          RET               Retour au Basic
│└►00          NOP               Pas d'opération
│  2A,0E,40    LD HL, 40 0E      Charger l'octet 40 0E dans HL
│  7E          LD A, HL          Caractère pointé par HL dans A
│  B7          OR A              Ou logique de A
│  C0          RET NZ            Si résultat n'est pas zéro retour au Basic
│  2B          DEC HL            Décrémentation de HL
│  22,0E,40    LD  40,0E ,HL     Charger HL à l'adresse 40 0E
│  3E,00       LD A,00           Caractère d'effacement dans A
│  D7          RST               Effacement de la raquette
│  3E,83       LD A,83           Caractère de la raquette dans A
│  D7          RST               Affichage de la raquette
│  06,FF       LD B,FF           Temporisation à droite
│┌►00          NOP               Pas d'opération
│└─10,FD       DJNZ,FD           Décrément de B et saut à -3 si non zéro
│  C9          RET               Retour au Basic
└─►00          NOP               Pas d'opération
   2A,0E,40    LD HL, 40,0E      Charger 40 0E dans HL
   2B,2B       DEC HL,DEC HL     Deux pas en arrière
   7E          LD A, HL          Caractère pointé par HL dans A
   B7          OR A              OU logique de A
   C0          RET NZ            Si le résultat n'est pas zéro retour Basic
   23          INC HL            Incrémenter HL
   22,0E,40    LD  40,0E ,HL     Charger HL à 40 0E
   3E,00       LD A,00           Caractère d'effacement dans A
   D7          RST               Effacement de la raquette
   2A,0E,40    LD HL, 40,0E      Charger le contenu de 40 0E dans HL
   2B,2B       DEC HL,DEC HL     Deux pas en arrière
   22,0E,40    LD  40,0E ,HL     Charger HL à 40 0E
   3E,83       LD A,83           Caractère de la raquette dans A
   D7          RST               Affichage de la raquette
   06,FF       LD B,FF           Temporisation à gauche
 ┌►00          NOP               Pas d'opération
 └─10,FD       DJNZ,FD           Décrément de B et saut à -3 si non zéro
   C9          RET               Retour au Basic
```

Le programme machine commence par l'établissement des limites des mouvements de la raquette matérialisés par deux barres verticales dans le haut de l'écran et place la raquette entre ces deux repères.

Le programme continue avec le test du clavier et suivant la touche pressée saute au déplacement vers la gauche ou vers la droite de la raquette. A chaque déplacement le programme retourne au Basic ce qui permet de ralentir les mouvements trop rapides.

Le programme chargeur qui suit contient le programme machine placé dans la chaîne de caractères A$ de la ligne 20.

Compte tenu du programme chargeur, un tel programme machine correspond à peu près au programme maximum qu'il est possible de placer dans 1 Kilo-octets de mémoire RAM.

```
   1 REM ......................
...............................
...............................
...........................
  10 LET A=16514
  20 LET A$="3E85D7061E3E00D710F
D3E05D72A0E4001F0FF09220E403E83D
700D5CDB8027D3CD128F7D5E5C1CDBD0
7D17E2A0E40FE21281DFE242801C9002
A0E407EB7C02B220E403E00D73E83D70
6FF0010FDC9002A0E402B2B7EB7C0232
20E403E00D72A0E402B2B220E403E83D
706FF0010FDC9"
  30 FOR B=1 TO LEN A$-1 STEP 2
  40 LET C=CODE A$(B)-28
  50 LET D=CODE A$(B+1)-28
  60 POKE A,16*C+D
  70 LET A=A+1
  80 NEXT B
```

Programme Basic utilisateur :

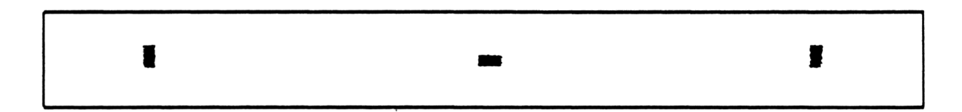
```
   1 REM Y▌NOT ▞2Y NOT ( CLEAR Y
▌NOT E:RND▘ LIST COPY ▒6:RNDY▄NO
T  STR$ LN █▘?WSGN C RUN STR$ FA
ST AT LN █▘SGN C1 RETURN 8C▘TAN
 E:RNDRNDY NOT Y▄NOT ▞ COPY  ( C
LEAR TAN  E:RNDFFRNDY NOT E:RNDP
F6:RNDY▄NOT ▞ COPY  ( CLEAR TAN
....
  20 RAND USR 16514
  30 RAND USR 16541
  40 GOTO 30
```

Le début du jeu

Un déplacement

Lorsque ce programme est lancé les deux repères s'affichent à droite et à gauche dans le haut de l'écran et la raquette se place entre ces deux repères et reste immobile en attendant que vous pressiez la touche 5 pour la diriger vers la gauche ou la touche 8 pour la diriger vers la droite.

L'exemple suivant montre l'affichage sur l'écran après le lancement du programme de la raquette.



On remarquera que ce programme Basic appelle une première fois le programme machine à son début à l'adresse 16514, puis par la suite à l'adresse 16541 sur le test du clavier qui contrôle les mouvements de la raquette.

LES COMMANDES ÉLECTRONIQUES

Dans le cours de cet ouvrage, nous nous sommes limités à des applications du langage machine qui concernaient le micro-ordinateur ZX 81 et l'écran du téléviseur. Cependant ce micro-ordinateur peut permettre des quantités d'autres applications.

Le connecteur qui est situé à la partie arrière du ZX 81, et sur lequel vient s'enficher le module d'extension mémoire permet aussi de connecter différents autres modules d'extension du micro-ordinateur.

Un de ces modules, la carte 8 entrées et 8 sorties programmables, est particulièrement intéressant. Ce module permet de réaliser une quantité de commandes électroniques. Cette carte est disponible chez les distributeurs du micro-ordinateur ZX 81.

Avec cette carte 8 E/S il est possible de transformer un ZX 81 en un automate programmable.

Grâce à cette carte 8 E/S nous pouvons utiliser le programme 31 (horloge numérique) pour commander un appareil électrique à une heure déterminée. Par exemple mettre en marche le four électrique à 11 heures de manière à trouver le déjeuner cuit à 12 heures.

Des quantités d'autres applications de la carte 8 E/S peuvent être envisagées, depuis le système d'alarme antivol perfectionné jusqu'à la commande d'une machine-outil automatique.

La carte 8 E/S est basée sur l'utilisation d'un circuit intégré PIO (Parallel Input Output), qui permet de disposer de ports d'entrée et de sortie.

Les ports sont des sorties (ou des entrées) de registres 8 bits, semblables à des cases mémoire, qui permettent au microprocesseur du ZX 81 de communiquer avec l'extérieur.

Il est possible avec cette carte 8 E/S de brancher les 8 sorties pour commander 8 appareils électriques différents.

Si cela est nécessaire, on peut brancher plusieurs cartes 8 E/S sur un ZX 81 pour augmenter le nombre des ports d'entrée et de sortie.

Le port d'entrée peut recevoir 8 commandes extérieures sur ses 8 entrées ; ces commandes peuvent modifier le programme en cours.

Pour mieux comprendre le fonctionnement des ports de la carte 8 E/S, il est utile de revoir le chapitre sur la numération binaire.

Nous allons donner quelques exemples montrant l'utilisation de la carte 8 E/S dans l'application des commandes électroniques.



COMMANDE FEUX DE CROISEMENT TRICOLORES (1 K)

Le programme va nous permettre de commander successivement les feux tricolores réglant la circulation à un carrefour.

Le programme en langage machine permet d'utiliser la carte 8 E/S.

Ce programme comporte 2 parties, la première commande le port d'entrée, la deuxième le port de sortie.

Lorsque la partie commande du port de sortie est mise en service par USR 16528 l'octet qui se trouve à l'adresse 16527 apparaît sur les bornes de sortie de la carte 8 E/S.

Lorsque la partie commande du port d'entrée est mise en service par USR 16534 les signaux reçus sur les 8 bornes d'entrée se trouvent reportés dans l'octet situé à l'adresse 16527. Cette partie ne sera pas utilisée dans le programme qui suit.

Voici le programme machine qui met en service la carte 8 E/S.

	Hexadécimal	**Assembleur**	
16528	3A,8F,40	LD A, 40 8F	*Le contenu de l'adresse 16527 dans A*
	D3,3F	OUT A	*Commande de sortie*
	C9	RET	*Retour au Basic*
16534	DB,3F	IN A	*Commande d'entrée*
	32,8F,40	LD 40 8F ,A	*A dans l'adresse 16527*
	C9	RET	*Retour au Basic*

Programme chargeur :

```
    1 REM ........................
.......
   10 LET A=16527
   20 LET A$="003A8F40D33FC9DB3F3
28F40C9"
   30 FOR B=1 TO LEN A$-1 STEP 2
   40 LET C=CODE A$(B)-28
   50 LET D=CODE A$(B+1)-28
   60 POKE A,16*C+D
   70 LET A=A+1
   80 NEXT B
```

Après une première utilisation du programme chargeur, la ligne 1 de ce programme contenant le programme machine servira de sous-programme à la commande des feux tricolores.

Les 8 sorties de la carte 8 E/S peuvent commander directement des voyants ou des petits moteurs électriques à condition que ces appareils ne consomment pas plus de 1 ampère sous 30 volts en courant continu.

Lorsqu'on désire commander des appareils d'une plus grande puissance, les sorties de la carte 8 E/S commandent des relais qui permettent de contrôler toutes sortes d'appareils électriques.

Le programme qui suit commande successivement les 3 feux de signalisation.

La sortie 1 commande le feu vert, la sortie 2 le feu orange et la sortie 3 le feu rouge.

Il est possible de régler le temps d'allumage de chaque feu en modifiant le temps de pause correspondant, une seconde équivaut à PAUSE 50.

Voici le programme de feux tricolores :

```
   1 REM ............. ■U■RNDPEEK
ZTAN <=ZM■RNDTAN ....
100 LET A=16527
110 POKE A,1
115 GOSUB 200
120 PAUSE 500
130 POKE A,2
135 GOSUB 200
140 PAUSE 200
150 POKE A,4
155 GOSUB 200
160 PAUSE 500
190 GOTO 110
200 RAND USR (A+1)
210 RETURN
```

L'orange sera plus court que le rouge et le vert



PREMIERE COMMANDE DE TRAIN ELECTRIQUE (1 K)

Le ZX 81 équipé d'une carte 8 E/S est le moyen idéal pour réaliser des commandes programmées des trains électriques modèles réduits. Une infinité de programmes allant du plus simple au plus évolué peuvent être ainsi conçus.

Le programme qui suit entre dans la catégorie des plus simples.

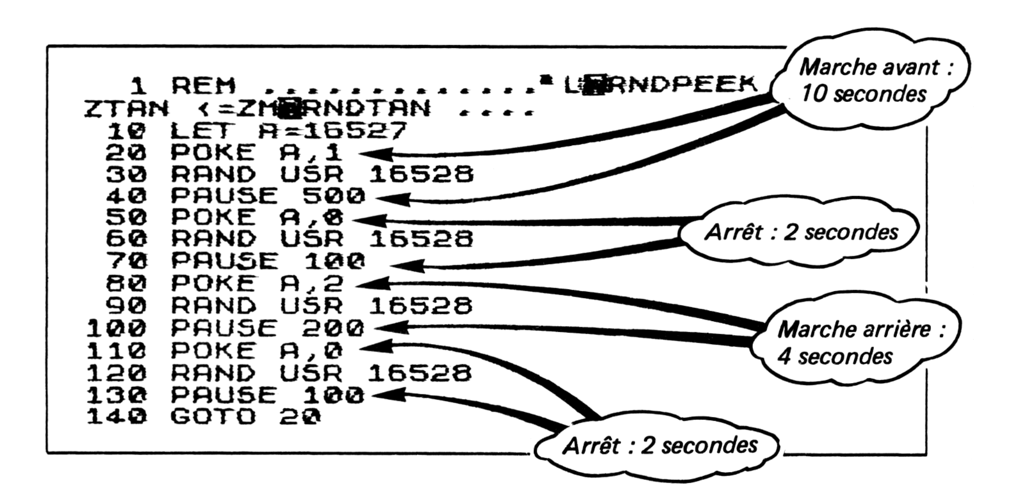
La sortie 1 de la carte 8 E/S commande la marche avant du train, la sortie 2 la marche arrière et le train est arrêté lorsque toutes les sorties sont à zéro.

Ce programme répète constamment le cycle suivant, marchant avant, arrêt, marche arrière, arrêt et retour au début du cycle.

Le temps de chacune des phases du cycle peut être réglé comme précédemment.

Le programme machine de mise en service de la carte E/S est toujours le même.

Voici le programme utilisateur :





DEUXIEME COMMANDE DE TRAIN ELECTRIQUE (1 K)

Dans le premier programme de train électrique, le temps de chaque phase du cycle de fonctionnement de la marche du train était déterminé par des temporisations programmées.

Dans ce deuxième programme ces temps seront déterminés par le signal qui sera appliqué sur une des 8 bornes d'entrée.

Lorsqu'aucun signal n'est appliqué sur les 8 bornes d'entrée de la carte 8 E/S, ces entrées sont au niveau 1 et la lecture de ces entrées donne l'octet FF ou 255 en décimal. Si on met une de ces 8 entrées au niveau zéro en la reliant à la borne moins de l'alimentation du ZX 81 la lecture des entrées sera modifiée en conséquence.

Lorsque les 8 entrées de la carte 8 E/S sont reliées au pôle négatif de l'alimentation, la lecture des entrées donnera 0.

Dans le programme qui suit, nous utilisons un interrupteur pour relier la borne 8 des entrées au moins de l'alimentation. La lecture des entrées sera donc en décimal 255 lorsque l'interrupteur sera ouvert et 127 lorsque l'interrupteur sera fermé.

Le programme sera lancé avec l'interrupteur ouvert et le train partira en marche avant, il s'arrêtera lorsque l'interrupteur se fermera, pour repartir en marche arrière à l'ouverture de celui-ci, une nouvelle fermeture de l'interrupteur arrête le train qui repart en marche avant à l'ouverture de celui-ci.

Voici ce deuxième programme :

```
   1 REM ............... U■RNDPEEK
 ZTAN <=ZM■RNDTAN ....
  10 LET A=16527
  20 POKE A,1
  30 RAND USR (A+1)
  40 RAND USR (A+7)
  50 IF PEEK A<>127 THEN GOTO 20
  60 POKE A,0
  70 RAND USR (A+1)
  80 RAND USR (A+7)
  90 IF PEEK A=127 THEN GOTO 60
 100 POKE A,2
 110 RAND USR (A+1)
```

Marche avant

Lecture des entrées

```
 120 RAND USR (A+7)
 130 IF PEEK A<>127 THEN GOTO 10
0
 140 POKE A,0
 150 RAND USR (A+1)
 160 RAND USR (A+7)
 170 IF PEEK A=127 THEN GOTO 140
 180 GOTO 20
```

Lorsque ces programmes utilisant la carte 8 E/S sont en service, l'écran du téléviseur est devenu inutile et celui-ci peut être supprimé sans inconvénient, à moins que vous ne vouliez l'utiliser comme tableau de bord en ajoutant au programme Basic quelques instructions d'affichage...

10 PROGRAMMES JEUX

Nous allons voir quelques programmes de jeux dans lesquels le langage machine est utilisé.

Le plus souvent celui-ci servira à obtenir des mouvements plus rapides sur l'écran que ceux permis par le langage Basic.

La plupart de ces programmes nécessitent l'utilisation du module d'extension 16 K car si l'on désire un affichage complet de l'écran, il faut plus de 700 octets pour ce seul affichage, et il ne resterait pratiquement rien pour le programme si l'on n'utilisait pas l'extension mémoire.

Les programmes qui suivent pourront ne comporter que quelques octets en langage machine ou au contraire être constitués en totalité par le programme en langage machine.

Ces programmes ont été choisis de manière à constituer un éventail d'exemples de l'utilisation du langage machine avec le micro-ordinateur ZX 81.

Au début de chaque programme figure la partie en langage machine du microprocesseur Z 80, seuls les codes hexadécimaux de ce langage sont donnés, sans la correspondance en langage assembleur de ces codes qui figurait dans les chapitres précédents.

Il est facile de retrouver cette information en se reportant aux chapitres précédents.

51 RALLYE AUTOMOBILE (1 K)

Ce programme va vous permettre de piloter un véhicule sur une route encombrée d'obstacles.

Le but de ce jeu est de parcourir le maximum de kilomètres sans avoir d'accidents.

Pour guider votre véhicule vous disposez de la touche 5 qui le dirige vers la gauche et de la touche 8 qui le dirige vers la droite. Les flèches gravées sur ces touches indiquent le sens du guidage.

Voici les 8 octets qui constituent la partie du programme en langage machine.

```
2A 0E 40 4E 06 00 C9 00
```

Le programme chargeur qui suit mettra ce programme machine dans l'instruction REM de la ligne 1.

```
 1 REM ..........
10 LET A=16514
20 LET A$="2A0E404E0600C900"
30 FOR B=1 TO LEN A$-1 STEP 2
40 LET C=CODE A$(B)-28
50 LET D=CODE A$(B+1)-28
60 POKE A,16*C+D
70 LET A=A+1
80 NEXT B
```

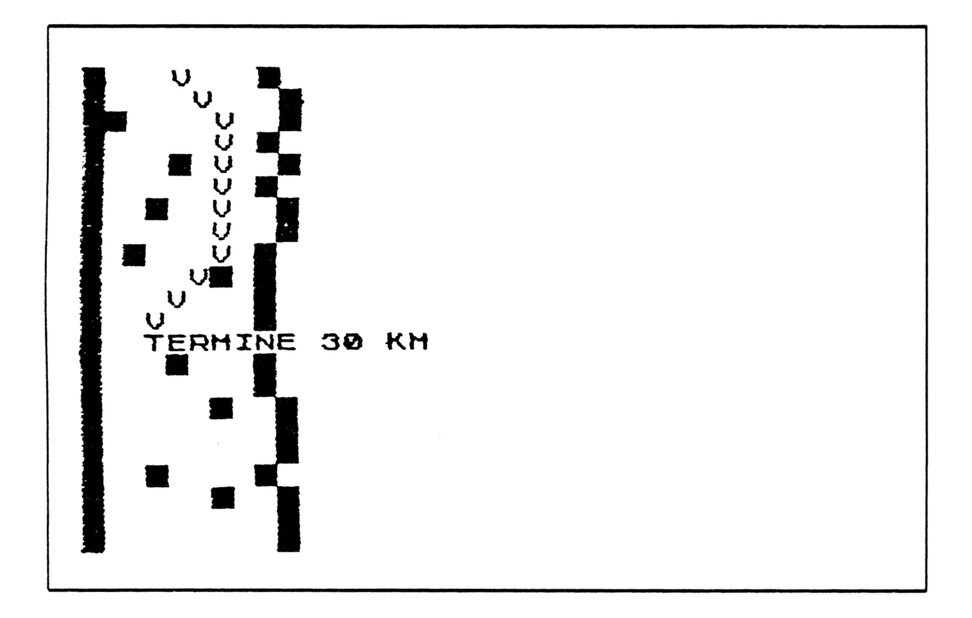
Lorsque ce programme chargeur est lancé, le programme machine contenu dans la chaîne de caractères A$ passe dans l'instruction REM de la ligne 1.

Les autres lignes du programme chargeur sont supprimées et le programme définitif est écrit.

Lorsque le programme est lancé, vous vous retrouvez pilotant votre véhicule au milieu des obstacles qui jalonnent la route.

```
   1 REM E:RND?▟ TAN
   5 CLS
   6 LET A=3
  10 LET S=0
  15 LET X=16514
  16 LET L=INT (RND*100)
  20 PRINT AT 21,0;"█";TAB 8+INT
 (RND*2);"█"
  25 IF L<40 THEN PRINT AT 21,IN
T (RND*8);"█"
  30 PRINT AT 12,A;
  40 IF USR X=128 THEN GOTO VAL
"150"
  50 PRINT AT 12,A;"V"
  60 LET A=A+(INKEY$="8" AND A<1
5)-(INKEY$="5" AND A>0)
  70 LET S=S+1
  80 SCROLL
  90 GOTO 16
 150 PRINT "TERMINE ";S;" KM"
```

L'exemple suivant montre la fin du jeu lorsque le véhicule vient de heurter un obstacle.



52 LE JEU DE LA VIE (1 K)

Le programme qui suit montre une application classique des possiblités de simulation des ordinateurs.

Le jeu de la vie montre l'évolution dans le temps d'une colonie de bactéries.

Le jeu consiste à placer au départ quelques bactéries et à observer l'évolution de cette colonie une fois le programme lancé.

Les bactéries peuvent se multiplier ou au contraire disparaître progressivement. Elles peuvent mourir par étouffement lorsqu'elles ont quatre voisines ou plus.

Elles peuvent mourir par isolement lorsqu'elles n'ont pas de voisine ou bien une seule voisine.

Une bactérie en contact avec deux voisines survit.
Trois bactéries voisines donnent naissance à une nouvelle bactérie.

Voici le programme machine qui comporte 153 octets.

```
01 00 00 3E 00 B8 20 02 06 10
B9 20 02 0E 10 3E 11 B8 20 02
06 01 B9 20 02 0E 01 26 00 69
CB 25 CB 25 CB 25 CB 25 CB 14
59 16 00 19 58 19 ED 5B 0C 40
19 11 01 00 19 C9 01 10 10 CD
85 40 7E FE 34 38 2D 0D CD 85
40 34 0C CD 85 40 34 0D CD 85
40 34 0C CD 85 40 34 05 CD 85
40 34 05 CD 85 40 34 0D CD 65
40 34 0D CD 85 40 34 04 0C CD
85 40 10 C9 06 10 0D 20 C4 01
10 10 CD 85 40 7E FE 03 28 08
FE 36 28 04 FE 37 20 04 3E 34
18 02 3E 00 77 10 E7 06 10 0C
20 E2 C9
```

Le programme chargeur qui suit va placer le langage machine dans l'instruction REM de sa ligne 1.

Vous remarquerez que la chaîne de caractères A$ contenant le langage machine a été divisée en quatre parties pour faciliter les corrections en cas d'erreur.

```
   1 REM ........................
..................................
..................................
..................................
..................................
......
  10 LET A=16514
  15 GOTO 200
  20 LET L=0
  30 FOR B=1 TO LEN A$-1 STEP 2
  40 LET C=CODE A$(B)-28
  50 LET D=CODE A$(B+1)-28
  60 POKE A,16*C+D
  70 LET A=A+1
  80 NEXT B
  90 LET L=L+1
 100 IF L=1 THEN GOTO 220
 110 IF L=2 THEN GOTO 240
 120 IF L=3 THEN GOTO 260
 130 IF L=4 THEN STOP
 200 LET A$="0100003E00B82002061
0B920020E103E11B820020601B920020
E01260069CB25CB25CB25CB25CB14"
 210 GOTO 30
 220 LET A$="5916001958 19ED5B0C4
019110100 19C9011010CD85407EFE343
82D0DCD854034 0CCD8540340DCD85"
 230 GOTO 30
 240 LET A$="40340CCD85403405CD8
5403405CD8540340DCD8540340DCD854
034040CCD854010C906100D20C401"
 250 GOTO 30
 260 LET A$="1010CD85407EFE03280
8FE362804FE3720043E3418023E00771
0E706100D20E2C9"
 270 GOTO 30
```

Lorsque le programme chargeur est lancé, le programme machine est contenu dans l'instruction REM de la ligne 1, qui devient la première ligne du programme définitif.

Le programme du jeu de la vie est donné ci-dessous ; lorsque ce programme est lancé, le micro-ordinateur vous demande de positionner la colonie d'origine. Pour cela vous devez entrer successivement chaque bactérie à l'aide de ses coordonnées, « HG » par exemple.

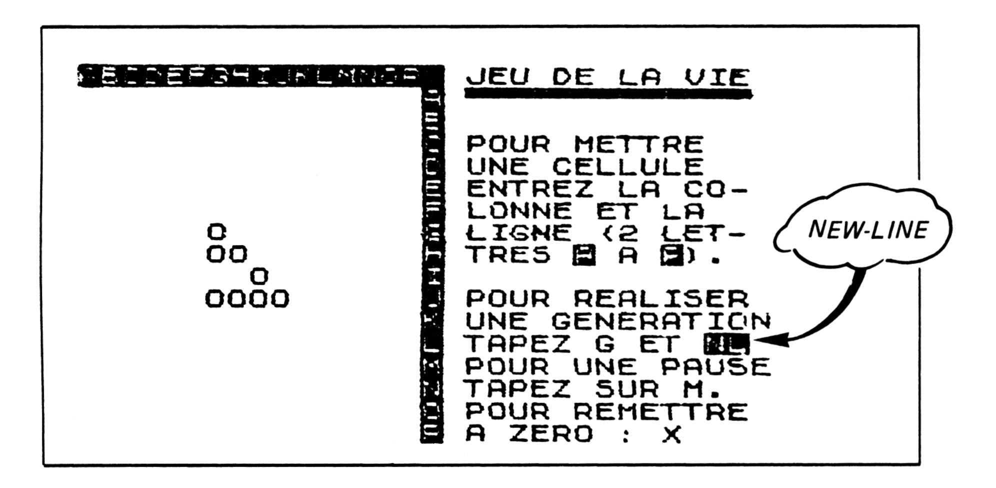
Voici trois exemples de colonies dont l'évolution est intéressante, mais vous en trouverez des centaines d'autres.

1 = HH,HI,HJ,II.
2 = HH,GG,II,JJ,IJ,HJ,GJ,GH.
3 = HH,II,JJ,HI,HJ.

```
 100 REM "  Y ▙4 "▞(▙4 ": (Y)▙4 "▀▜
4 ":"A ?ACS 9ACS 9ACS 9ACS 9ACS =
?- ;?; GOSUB ?£RND;)" ;TAN " ((LN
 ▌RNDLN ▌RNDO▖LN ▌RNDO£LN ▌RNDO£
LN ▌RNDO▌LN ▌RNDO▌LN ▌RNDO$LN ▌R
NDO$LN ▌RNDO▖£LN ▌RND(TAN ▞"($4CO
DE " ((LN ▌RNDQC▖ RETURN R4▖YO/"Y
 ?( SCROLL ▞"($4 LLIST TAN

 110 POKE 16388,0
 120 POKE 16389,0
 130 CLS
 140 PRINT AT 1,18;"JEU DE LA VI
E";TAB 18;"████████████";
 150 PRINT AT 4,18;"POUR METTRE"
 160 PRINT AT 5,18;"UNE CELLULE"
 170 PRINT AT 6,18;"ENTREZ LA CO
-"
 180 PRINT AT 7,18;"LONNE ET LA"
 190 PRINT AT 8,18;"LIGNE (2 LET
-"
 200 PRINT AT 9,18;"TRES A A P).
"
 210 PRINT AT 11,18;"POUR REALIS
ER";TAB 18;"UNE GENERATION";TAB
18;"TAPEZ G ET NL";TAB 18;"POUR
UNE PAUSE";TAB 18;"TAPEZ SUR M."
 220 PRINT AT 16,18;"POUR REMETT
RE";TAB 18;"A ZERO : X"
 420 LET A$=""
 430 LET C=0
 440 DIM T$(16,16)
 500 PRINT AT 1,0;"ABCDEFGHIJKLM
NOP"
 520 FOR K=2 TO 17
 530 PRINT AT K,16;CHR$ (K+164)
 540 NEXT K
 600 INPUT A$
 620 IF A$="G" AND C=0 THEN GOTO
800
 621 IF A$="G" THEN GOTO 700
 625 IF CODE A$<38 OR CODE A$>53
 OR CODE A$(2)<38 OR CODE A$(2)>
53 THEN GOTO 600
 630 PRINT AT CODE A$(2)-36,CODE
 A$-38;"O"
 640 LET T$(CODE A$(2)-37,CODE A
$-37)="O"
 680 GOTO 600
 700 LET C=USR 16570
 710 PAUSE 60
 712 POKE 16437,255
 715 IF INKEY$="X" THEN GOTO 130
 720 IF INKEY$<>"M" THEN GOTO 70
0
 730 GOTO 600
 800 CLS
 810 PRINT AT 0,15;" "
 820 FOR K=2 TO 17
 830 PRINT AT K,0;T$(K-1)
 840 NEXT K
 850 GOTO 700
```

L'exemple qui suit montre l'image sur l'écran lorsque le programme du jeu de la vie est lancé et que la colonie numéro 2 a été positionnée.



Tel que nous venons de le voir, le programme du jeu de la vie est étudié pour fonctionner sur un ZX 81 équipé d'un module d'extension mémoire.

Il est possible de simplifier la partie Basic du programe tout en conservant le même programme machine de façon à permettre au programme du jeu de la vie de fonctionner sur un micro-ordinateur ZX 81 n'ayant que 1 K de mémoire.

53 JEU DE LA VIE (1 K)

Le programme du jeu de la vie qui suit est sembable au précédent mais est simplifié pour être utilisé avec la version 1 K du micro-ordinateur ZX 81.

Le programme machine reste inchangé et comporte toujours les mêmes 153 octets.

```
01  00  00  3E  00  B8  20  02  06  10
B9  20  02  0E  10  3E  11  B8  20  02
06  01  B9  20  02  0E  01  26  00  69
CB  25  CB  25  CB  25  CB  25  CB  14
59  16  00  19  58  19  ED  5B  0C  40
19  11  01  00  19  C9  01  10  10  CD
85  40  7E  FE  34  38  2D  0D  CD  85
40  34  04  CD  85  40  34  0C  CD  85
40  34  0C  CD  85  40  34  05  CD  85
40  34  05  CD  85  40  34  0D  CD  85
40  34  0D  CD  65  40  34  04  0C  CD
85  40  10  C9  06  10  0D  20  C4  01
10  10  CD  85  40  7E  FE  03  28  08
FE  36  28  04  FE  37  20  04  3E  34
18  02  3E  00  77  10  E7  06  10  0D
20  E2  C9
```

Pour charger ce programme machine nous ne pouvons utiliser le même programme chargeur que celui fait pour la version 16 K. Ce programme chargeur excèderait le kilo-octet de mémoire de la version de base du ZX 81.

Nous utilisons donc un programme chargeur plus réduit qui nous permettra de placer les octets du langage machine un par un dans la mémoire RAM.

Il faudra faire attention lors de cette opération car les vérification et les corrections permises par l'utilisation de la chaîne de caractères A$ ne sont plus possibles.

```
   1 REM .....................
................................
................................
................................
................................
.......
  10 LET A=16514
  20 INPUT A$
  30 CLS
  40 PRINT A;"  ";A$
  50 LET B=((CODE A$(1)-28)*16)+
(CODE A$(2)-28)
  60 POKE A,B
  70 LET A=A+1
  80 GOTO 20
```

Lorsque le programme machine a été entièrement placé dans l'instruction REM de la ligne 1, les autres lignes du programme chargeur sont supprimées.

Le programme définitif est le suivant ;

```
   1 REM ▝  Y ▒4 ▜▘(▒4 ▝: (Y)▒4 ▜▀▒
4 ▝:▝A ?ACS 9ACS 9ACS. 9ACS 9ACS =
?- ;?; GOSUB ?£RND;)▝ ;TAN ▝ ((LN
 ▐RNDLN ▐RNDO▖LN ▐RNDO£LN ▐RNDO£
LN ▐RNDO▌LN ▐RNDO▌LN ▐RNDO$LN ▐R
NDO$LN ▐RNDO▖£LN ▐RND(TAN ▞($4CO
DE ▝ ((LN ▐RNDQC▖ RETURN R4▖YO/ ▀Y
 ?( SCROLL ▞($4 LLIST TAN ...
 100 CLS
 120 LET A$=""
 130 LET C=0
 200 PRINT AT 0,15;" "
 220 FOR K=2 TO 17
 230 PRINT AT K,15;" "
 240 NEXT K
 300 INPUT A$
 320 IF A$="G" THEN GOTO 400
 330 PRINT AT CODE A$(2)-36,CODE
A$-38;"O"
 340 GOTO 300
 400 RAND USR 16570
 410 PAUSE 50
 420 POKE 16437,255
 430 GOTO 400
```

Ce programme est une version simplifiée du programme 16 K du jeu de la vie.

Les trois exemples d'implantation des colonies de base que nous avons vu avec la version 16 K peuvent être utilisés de la même manière avec le jeu de la vie 1 K.

L'exemple qui suit montre l'image sur l'écran lorsque la colonie créée avec l'exemple 2 est dans sa phase d'expansion.

```
         O
        OOO
       OOOOO
      O     O
     OO  O  OO
    OOO O O OOO
     OO  O  OO
      O     O
       OOOOO
        OOO
         O
```

Examen des octets en mémoire

Il est souvent intéressant de retrouver la liste des octets placés dans une instruction REM et constituant le langage machine d'un programme.

Le programme suivant permet l'examen du langage machine contenu dans l'instruction REM du programme du jeu de la vie.

```
   1 REM ▀   Y ▒4 ▀▀(▒4 ▀: (Y)▒4 ▀▀▀▒
4 ▀:▀A ?ACS 9ACS 9ACS 9ACS 9ACS =
?- ;?; GOSUB ?£RND;)▀  ;TAN ▀ ((LN
 ▌RNDLN ▌RNDO▄LN ▌RNDO£LN ▌RNDO£
LN ▌RNDO▌LN ▌RNDO▌LN ▌RNDO$LN ▌R
NDO$LN ▌RNDO▄£LN ▌RND(TAN ▄▀($4CO
DE ▀ ((LN ▌RNDQC▄ RETURN R4▄YO/▀Y
 ?( SCROLL ▄▀($4 LLIST TAN ...
  10 REM EXAMEN DES OCTETS MEMOI
RE
  20 LET A=16514
  30 FOR F=1 TO 10
  40 LET B=PEEK A
  50 LET C=INT (B/16)
  60 LET D=B-(C*16)
  70 PRINT CHR$ (C+28);CHR$ (D+2
8);" ";
  80 LET A=A+1
  90 IF A>=16670 THEN STOP
 100 NEXT F
 110 PRINT
 120 GOTO 30
```

Lorsque ce programme est lancé, on voit s'inscrire sur l'écran les 157 octets qui constituent le programme machine du jeu de la vie en code hexadécimal.

Mais ce programme peut être utilisé pour examiner d'autres zones de la mémoire. Il suffit de modifier l'adresse de début de la zone examinée à la ligne 20 et celle de la fin de cette zone à la ligne 90.



54 LES ENVAHISSEURS (16 K)

Les envahisseurs attaquent et vous devez les détruire à l'aide de votre canon laser.

Vous devez détruire le maximum d'envahisseurs avant que vos réserves d'énergie soit épuisées.

Voici les 75 octets qui constituent le programme machine de ce jeu.

```
2A 0E 40 06 11 36 05 0E 21 2B
0D 20 FC 5E 1C 1D 20 22 36 05
0E 21 23 0D 20 FC 36 00 0E 21
2B 0D 20 FC 0E FE 0D 20 FD 0E
FE 0D 20 FD 10 D7 36 00 01 00
00 C9 2B 2B 2B 0E 05 23 36 00
0D 20 FA 0E 1F 23 0D 20 FC 36
00 01 0C 08 C9
```

Le programme chargeur suivant fait passer ce programme machine dans l'instruction REM de la ligne 1 de ce programme chargeur.

```
   1 REM ......................
................................
................................
..............................
  10 LET A=16514
  20 LET A$="2A0E40061136050E212
B0D20FC5E1C1D2022360 50E21230D20F
C36000E212B0D20FC0EFE0D20FD0EFE0
D20FD10D7360001000 0C92B2B2B0E052
336000D20FA0E1F230D20FC360 0010C0
8C9"
  30 FOR B=1 TO LEN A$-1 STEP 2
  40 LET C=CODE A$(B)-28
  50 LET D=CODE A$(B+1)-28
  60 POKE A,16*C+D
  70 LET A=A+1
  80 NEXT B
```

Une fois le programme chargeur lancé, le programme définitif peut être établi en utilisant, pour débuter celui-ci, la ligne 1 qui contient le programme ou langage machine.

```
   1 REM E:RND▘)Q▌:5F$4 UNPLOT ?
01460▌:57$4 UNPLOT Q :5F$4 UNPLO
T : RETURN $4 CLEAR : RETURN $4
CLEAR (NOT Q ▘  TAN FFF:▌7Q $4 I
F :37$4 UNPLOT Q ▘£▌TAN ........
................................
.........................
   5 LET E$=" ** LES ENVAHISSEUR
S **"
  10 PRINT E$
```

```
   11 PRINT
   12 PRINT " VOUS DEVEZ DETRUIRE
 LE MAXIMUM  D ENVAHISSEURS AVAN
T L EPUISE- MENT DE VOS RESERVES
 D ENERGIE"
   13 PRINT AT 10,0;"   5- A GAUC
HE <--"
   14 PRINT "   8- A DROITE -->"
   15 PRINT "   0- TIR LASER"
   16 PAUSE 3600
   17 CLS
   50 LET HS=0
   60 LET S=0
   65 PRINT E$
   70 LET A=10
   75 LET Q=0
   80 LET L=0
   90 LET B=INT (RND*28)
  100 PRINT AT 21,0;"RECORD=";HS
  110 PRINT AT 4,B;" ▦ "
  120 LET A=A+(INKEY$="8" AND A<2
9)-(INKEY$="5" AND A>2)
  130 PRINT AT 19,A-2;" ▄█▄ ";AT 1
8,A;
  140 IF INKEY$="0" THEN LET L=US
R 16514
  150 IF INKEY$="0" THEN LET Q=Q+
2
  160 IF L<>0 THEN LET S=S+5
  170 IF Q>200 THEN GOSUB 1000
  190 IF RND>.92 THEN GOSUB 700
  195 PRINT AT 21,15;"SCORE ";S
  200 GOTO 80
  700 PRINT AT 4,0;
  701 FOR Z=26 TO 0 STEP -2
  704 PRINT AT 2,Z;"▄███▄  "
  705 LET A=A+(INKEY$="8" AND A<2
9)-(INKEY$="5" AND A>2)
  706 PRINT AT 19,A-2;" ▄█▄ ";AT 1
8,A;
  707 IF INKEY$="0" THEN LET L=US
R 16514
  709 IF INKEY$="0" THEN LET Q=Q+
5
  710 IF L<>0 THEN LET S=S+15
  711 IF L<>0 THEN PRINT AT 2,Z;"
"
  712 IF L<>0 THEN RETURN
  720 NEXT Z
  721 PRINT AT 2,0;"        "
  722 RETURN
 1000 PRINT AT 0,0;" TERMINE   VO
US AVEZ  DETRUIT   ";S;" ENVAHIS
SEURS"
 1005 IF S>HS THEN LET HS=S
 1010 PRINT "RECORD= ";HS
 1020 PAUSE 600
 1030 CLS
 1040 GOTO 60
 2000 SAVE "■"
 2100 RUN
```

Lorsque ce programme est lancé, les règles du jeu sont affichées sur l'écran pendant environ 30 secondes, pour commencer le jeu plus tôt, il suffit de presser sur une touche.

```
** LES ENVAHISSEURS **

VOUS DEVEZ DETRUIRE LE MAXIMUM
 D ENVAHISSEURS AVANT L EPUISE-
MENT DE VOS RESERVES D ENERGIE



    5- A GAUCHE <--
    8- A DROITE -->
    Ø- TIR LASER
```

Cet affichage indique les trois touches à presser dans le cours du jeu. L'exemple qui suit montre une phase d'une partie en cours.

```
** LES ENVAHISSEURS **









RECORD=495        SCORE 55
```

Chaque fois que vous détruisez un envahisseur vous marquez 5 points. De temps en temps des soucoupes volantes traversent l'écran, leur destruction vous fait gagner 15 points.

Pour sauver ce programme sur cassette, faites GOTO 2ØØØ. Lorsque par la suite ce programme sera chargé il démarrera dès la fin du chargement.

55 ZX ARTISTE (16 K)

Le micro-ordinateur ZX 81 possède un talent artistique abstrait qui ne demande qu'à s'exprimer.

Le programme suivant va le demontrer.

La partie langage machine de ce programme comporte les 72 octets suivants.

```
2A 0C 40 0E 16 06 10 54 5D 3E
21 13 3D FE 00 20 FA 23 1B 7E
12 10 FA 3E 11 23 3D FE 00 20
FA 02 20 E3 2A 0C 40 0E 0B 54
5D 14 14 3E B5 13 3D FE 00 20
FA 06 20 23 13 7E 12 10 FA 23
3E 41 1B 3D FE 00 20 FA 0D 20
EC C9
```

Le programme chargeur qui suit fera passer le langage machine dans l'instruction REM.

```
   1 REM ........................
.................................
.................................
.................................
.
  10 LET A=16514
  20 LET A$="2A0C400E16061054 5D3
E21133DFE0020FA231B7E1210FA3E112
33DFE0020FA0220E32A0C400E0B545D1
4143EB5133DFE0020FA062023137E121
0FA233E411B3DFE0020FA0D20ECC9"
  30 FOR B=1 TO LEN A$-1 STEP 2
  40 LET C=CODE A$(B)-28
  50 LET D=CODE A$(B+1)-28
  60 POKE A,16*C+D
  70 LET A=A+1
  80 NEXT B
```

Lorsque le programme en langage machine a été placé dans l'instruction REM, le programme définitif peut être écrit à la suite de cette instruction REM.

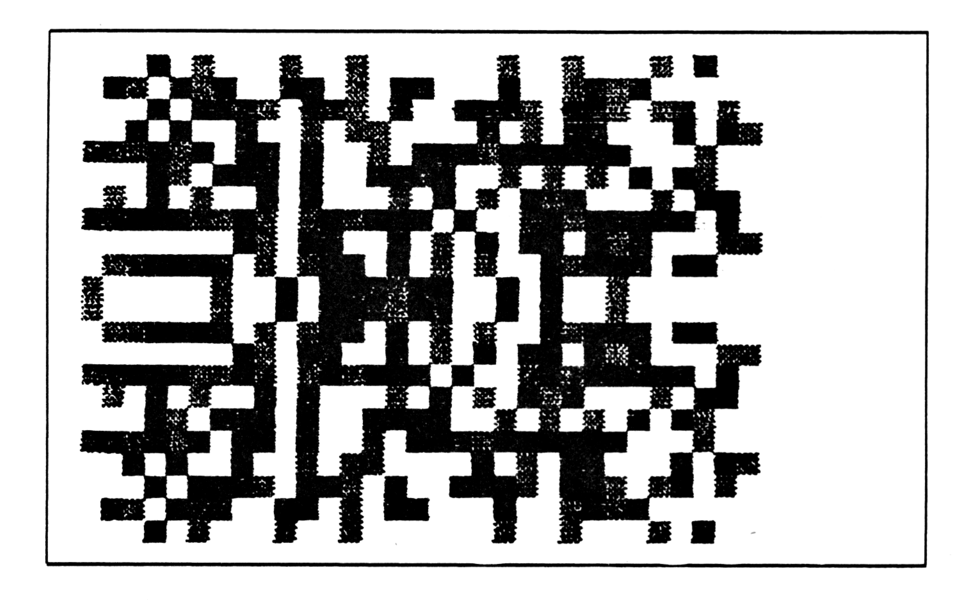
Voici ce programme qui est relativement réduit. Cependant l'utilisation de l'extension 16 K reste indispensable à cause de l'affichage.

```
   1 REM E£RND:-▞(??Y5<X RETURN
 4 IF 7.7X RETURN  4 IF ▝4 STOP
E£RND:"??==Y▣<X RETURN  4 IF ▞47
<INKEY$.X RETURN  4 IF $4 GOTO T
AN ..............................
,................
   5 LET A=INT (RND*10)+1
  10 LET A$="███ ▒▒▒▒████▒▒"
  15 LET B$=A$(A)
  20 PRINT AT RND*10,RND*30;B$;A
T 10-(RND*10),30-(RND*30);B$
  25 REM ARTISTE ZX 81
  30 RAND USR 16514
  40 GOTO 5
```

Lorsque le programme est lancé, le micro-ordinateur choisit un des dix caractères de la chaîne A$ de la ligne 20 et affiche deux fois ce caractère dans le haut de l'écran. Immédiatement le programme en langage machine reproduit symétriquement cet affichage dans le bas de l'écran.

En changeant les caractères dans la chaîne A$ on obtient un autre type de dessin.

L'affichage sur l'écran change constamment lorsque ce programme est en fonctionnement. L'exemple suivant en montre un « instantané » :



56 LA BALLE AU VOL (16 K)

Un programme de jeu peut contenir une partie en langage machine destinée à accélérer le jeu.

Cette partie en langage machine peut se réduire à quelques octets ou au contraire constituer la totalité du programme.

Le programme de jeu de la balle au vol correspond à cette dernière possibilité.

Voici les 390 octets de ce programme en langage machine.

```
39 2E 32 2A 14 1E 1C 1C 1C 27 26
 31 31 38 14 1C 1C 1C 1C 08 08 0
8 08 08 81 82 08 08 84 07 08 08
08 08 08 2A 0C 40 11 D7 02 19 EB
 21 82 40 01 09 00 ED B0 1B ED 5
3 3C 40 E5 21 0A 00 19 EB 01 0A
00 E1 ED B0 1B ED 53 3E 40 3E 83
 CD 02 42 0E 14 3E 85 D7 3E 08 0
6 1E D7 10 FD 3E 05 D7 0D 20 F0
3E 03 CD 02 42 11 19 0A ED 53 41
 40 3A 34 40 CB 3F 06 7F B8 F2 0
1 41 67 78 D6 14 47 7C 18 F4 90
3C 57 1E 02 3E 01 BA 20 04 3E 14
 18 07 3E 14 BA 20 05 3E 15 32 2
C 41 3E 02 BB 20 04 3E 1C 18 07
3E 1F BB 20 05 3E 1D 32 2D 41 15
 1D CD E5 41 7E FE 08 28 1E 2A 3
E 40 7E FE 25 28 03 34 18 05 36
1C 2B 18 F3 3A 57 41 FE 5B 28 02
 D6 05 32 57 41 18 9A 36 80 06 0
0 0E 0F 0D 20 FD 10 F9 36 08 ED
53 43 40 CD BB 02 7C 2F 95 ED 5B
5B 41 40 FE 29 20 01 1D FE 19 20
 01 1C FE 21 20 01 15 FE 31 20 0
1 14 7A FE 13 20 01 15 FE 01 20
01 14 7B FE 1F 20 01 1D FE 03 20
 01 1C CD E5 41 ED 53 41 40 37 1
1 22 00 ED 52 EB 21 95 40 0E 04
C5 01 04 00 ED B0 EB 01 1D 00 09
 EB C1 0D 20 F0 ED 5B 43 40 2A 3
C 40 7E FE 14 20 10 06 04 2A 3C
40 36 1C 2B 10 FB 3E 00 32 57 41
 C9 FE 1C 28 04 35 C3 06 41 36 2
5 2B 18 DE 26 00 6A CB 25 CB 25
CB 25 CB 25 CB 14 CB 25 CB 14 06
 00 4A 09 4B 09 ED 4B 0C 40 09 C
9 06 20 D7 10 FD C9 1C 1C 1C 1C
```

Le programme chargeur qui suit placera ce programme en langage machine dans l'instruction REM au début du programme.

```
   1 REM ...........................
................................
................................
................................
................................
................................
................................
................................
................................
................................
................................
................................
................................
...
  10 LET A=16514
  15 LET L=200
  20 GOTO L
  30 FOR B=1 TO LEN A$-1 STEP 2
  40 LET C=CODE A$(B)-28
  50 LET D=CODE A$(B+1)-28
  60 POKE A,16*C+D
  70 LET A=A+1
  80 NEXT B
  90 LET L=L+20
 100 IF L=360 THEN STOP
 110 GOTO L
 200 LET A$="392E322A141E1C1C1C2
726313138141C1C1C1C08080808088818
20808840708080808082A0C4011D7021
9EB21824001090OED"
 210 GOTO 30
 220 LET A$="B01BED533C40E5210A0
019EB010A00E1EDB01BED533E403E83C
D02420E143E85D73E08061ED710FD3E0
5D70D20F03E03CD02"
 230 GOTO 30
 240 LET A$="4211190AED5341403A3
440CB3F067FB8F201416778D614477C1
8F4903C571E023E01BA20043E1418073
E14BA20053E15322C"
 250 GOTO 30
 260 LET A$="413E02BB20043E1C180
73E1FBB20053E1D322D41141DCDE5417
EFE08281E2A3E407EFE252803341805З
61C2B18F33A5741FE"
 270 GOTO 30
 280 LET A$="5B2802D6053257411189
A368006000E0F0D20FD10F93608ED534
340CDBB027C2F95ED5B4140FE2920011
DFE1920011CFE2120"
 290 GOTO 30
 300 LET A$="0115FE3120011147AFE1
320011SFE0120011147BFE1F20011DFE0
320011CCDE541ED5341403711 2200ED5
2EB2195400E04C501"
 310 GOTO 30
 320 LET A$="0400EDB0EB011D0009E
8C10D20F0ED5B43402A3C407EFE14201
006041A3C40361C2B10FB3E00325741C
9FE1C280435C30641"
 330 GOTO 30
```

```
340 LET A$="36252B18DE26006ACB2
5CB25CB25CB25CB14CB25CB1406004A0
94B09ED4B0C4009C90620D710FDC9"
350 GOTO 30
```

Dans ce programme chargeur la chaîne de caractères A$ qui contient les octets hexadécimaux a été divisée en plusieurs parties pour faciliter les corrections. Mais ce programme fonctionnerait tout aussi bien si la chaîne A$ était d'un seul bloc, comme dans les autres programmes que nous avons vus précédemment.

Lorsque le programme machine a été placé dans l'instruction REM, les lignes du programme chargeur devenues inutiles peuvent être supprimées pour être remplacées par les deux lignes 10 et 20 du programme définitif qui suit.

La ligne 10 peut même être supprimée sans inconvénient.

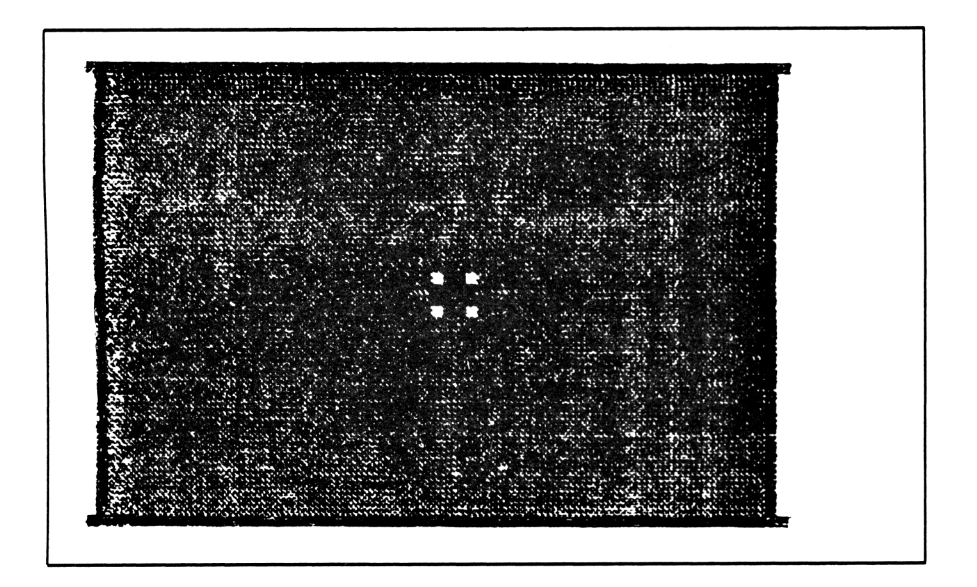
```
   1 REM TIME=2000BALLS=000
     E£RND)NOT  ; FOR 5 R
ND    GOSUB  . GOSUB ?WRND FAST
5  ; FOR     LPRINT GOSUB  . GOS
UB ?YRNDY LN  PI: =Y NOT Y  2NOT
( CLEAR Y NOT $4 LIST Y LN  PI);
   GOSUB ?INKEY$RNDUORNDACS Z
PAUSE  INKEY$??CHR$ =??/ POKE  W
?2  Y  4 Y=/ Y= 4 Y+MGINKEY$Y  4
Y0/ Y3 4 Y1MHINKEY$=1LN FAST INK
EY$YRND/ Q0F/ NEXT U?INKEY$ RETU
RN ?C CHR$  M?INKEY$/ Q  :?$4 C
LEAR ( RAND Q  GOSUB ??RNDLN   ?
J  GOSUB ?INKEY$RND RETURN D4 1
RETURN ;4 0 RETURN 54 + RETURN L
4 =? RETURN <4 + RETURN  4 =? RE
TURN 34 1 RETURN  4 0LN FAST INK
EY$ GOSUB ?INKEY$RNDR)6  GOSUB ?
 FOR 5 RND: VAL     GOSUB   FOR
 1   FOR AT $4 LIST GOSUB ??RNDE
WRND ,WRND00F( CLS Y M?INKEY$TAN
 RETURN 0C P? INKEY$Q9F/ THEN A
?ACS 9ACS 9ACS 9ACS 9ACS =ACS 9A
CS =  ?  ?  GOSUB ?£RND TAN  4NOT
 ( CLEAR TAN ....................

  10 REM BALLE AU VOL
  20 RAND USR 16549
```

Lorsque ce programme est lancé, une balle noire parcourt dans tous les sens l'écran devenu gris.

Le jeu consiste à intercepter le maximum de fois cette balle dans le temps

alloué en vous servant des touches 5 à 8. A chaque interception vous marquez un point, mais le jeu devient plus difficile car la balle change de trajectoire et la vitesse augmente.



57 LE CIRCUIT INFERNAL (16 K)

Le programme précédent était entièrement constitué par le langage machine, par contre dans le programme du circuit infernal, la partie en langage machine est très réduite. Elle ne comporte que les 20 octets suivants.

```
2A 0C 40 06 17 2B 23 7E FE 76
20 03 10 F8 C9 C6 80 77 18 F2
```

Voici le programme chargeur qui placera cette partie du programme en langage machine dans l'instruction REM.

```
   1 REM .......................
.....
  10 LET A=16514
  20 LET A$="2A0C40061728237EFE7
6200310F8C9C68077 18F2"
  30 FOR B=1 TO LEN A$-1 STEP 2
  40 LET C=CODE A$(B)-28
  50 LET D=CODE A$(B+1)-28
  60 POKE A,16*C+D
  70 LET A=A+1
  80 NEXT B
```

Lorsque la partie du programme en langage machine a été placée dans l'instruction REM, le programme définitif du circuit infernal peut être écrit à la suite de cette instruction REM.

Voici ce programme :

```
   1 REM E£RND▞*F7 SAVE TAN LEN
█?/ PAUSE ...........
   2 REM LE CIRCUIT INFERNAL
   4 LET H1=0
   5 LET S=0
   6 LET S1=0
   8 RAND
  10 PRINT "████████████████████
███████████"
  20 PRINT "█..............    ..
...........█"
  25 LET Q=0
  30 PRINT "█.████████████    ██
██████████.█"
  40 PRINT "█.█............    ..
..........█.█"
```

```
 50 PRINT "█.█.██████████    ██
████████.█.█"
 60 PRINT "█.█.█.........    ..
.......█.█.█"
 70 PRINT "█.█.█.████████    ██
██████.█.█.█"
 80 PRINT "█.█.█.█.......    ..
.....█.█.█.█"
 90 PRINT "█.█.█.█.████████████
████.█.█.█.█"
 100 FOR A=1 TO 4
 110 PRINT "█";TAB 8;"█
     █";TAB 31;"█"
 120 NEXT A
 130 PRINT "█.█.█.█.████████████
████.█.█.█.█"
 140 PRINT "█.█.█.█.......    ..
.....█.█.█.█"
 150 PRINT "█.█.█.████████    ██
██████.█.█.█"
 160 PRINT "█.█.█.........    ..
.......█.█.█"
 170 PRINT "█.█.██████████    ██
████████.█.█"
 180 PRINT "█.█............    ..
..........█.█"
 190 PRINT "█.████████████    ██
██████████.█"
 200 PRINT "█..............    ..
...........█"
 202 LET V=224
 205 LET G=14
 207 LET H=27
 209 LET V1=V
 210 PRINT "████████████████████
████████████"
 215 LET A1=PEEK 16396+256*PEEK
16397
 220 LET B1=0
 225 LET LA=1
 230 LET A=A1+678
 235 LET LB=1
 240 LET B=A1+299
 245 LET A2=0
 250 LET C=1
 257 LET V=V-1
 258 IF S>1 THEN RETURN
 260 LET D=-33
 270 IF A2=H THEN LET S=S+1
 280 IF PEEK (A+C)=128 THEN GOSU
B 400
 290 POKE A,G*((A2=H)+(A2=G))
 292 IF S=V1 THEN GOSUB 900
 295 IF INKEY$<>"" AND PEEK (A+C
)=0 THEN GOSUB 700
 297 REM
 298 REM
 300 LET A=A+C
 304 LET A2=PEEK A
 305 IF PEEK A=12 THEN GOTO 500
 310 POKE A,62
```

```
 320 IF PEEK (B+D)=128 THEN GOSU
B 450
 330 POKE B,B1
 335 IF B1=0 AND Q=0 AND LA<>LB
THEN GOSUB 800
 337 IF B1<>0 THEN LET Q=0
 340 LET B=B+D
 345 IF PEEK B=24 THEN GOTO 500
 350 LET B1=PEEK B
 360 POKE B,12
 370 GOTO 270
 400 LET X=0
 402 IF C=1 THEN LET X=-33
 405 IF X=-33 THEN GOTO 435
 410 IF C=-33 THEN LET X=-1
 415 IF X=-1 THEN GOTO 435
 420 IF C=-1 THEN LET X=33
 425 IF X=33 THEN GOTO 435
 430 IF C=33 THEN LET X=1
 435 LET C=X
 440 RETURN
 450 LET Y=0
 452 IF D=-33 THEN LET Y=1
 455 IF Y=-33 THEN GOTO 485
 460 IF D=1 THEN LET Y=33
 465 IF Y=33 THEN GOTO 485
 470 IF D=33 THEN LET Y=-1
 475 IF Y=-1 THEN GOTO 485
 480 IF D=-1 THEN LET Y=-33
 485 LET D=Y
 490 RETURN
 500 POKE A,23
 510 FOR M=1 TO 26
 520 RAND USR 16514
 530 NEXT M
 585 LET S=S+S1
 590 SLOW
 600 PRINT AT 9,9;"SCORE ";S
 605 IF H1<S THEN LET H1=S
 610 PRINT TAB 9;"HI-SCORE ";H1
 620 PAUSE 35000
 630 CLS
 640 GOTO 5
 700 LET A3=A
 705 LET A$=INKEY$
 710 LET A=A+(((INKEY$="8")-(INK
EY$="5"))*(ABS C=33)+((INKEY$="5
")-(INKEY$="7"))*33*(ABS C=1))*2
 720 IF A>A1+726 OR A<A1 OR PEEK
 A<>0 THEN LET A=A3
 730 IF A=A3 THEN RETURN
 740 LET L5=LA+(C=-1)*(A$="6")+(
C=1)*(A$="7")+(C=-33)*(A$="5")+(
C=33)*(A$="8")
 750 IF L5=LA THEN LET L5=LA-1
 755 LET LA=L5
 760 RETURN
 800 LET Q=1
 810 LET D1=D
 820 GOSUB 450
 830 LET D2=D
```

```
 840 LET D=D1
 850 LET W=LA-LB
 860 IF W>1 THEN LET W=1
 870 IF W<-1 THEN LET W=-1
 875 LET LB=LB+W
 880 LET B=B+W*D2*2
 890 RETURN
 900 LET S1=S1+S
 910 LET S=0
 920 LET G=H
 930 IF H<>G THEN GOTO 950
 940 LET H=14
 950 LET V1=V
 960 RETURN
1200 SAVE "■"
1210 RUN
```

Lorsque le programme du circuit infernal est lancé, vous êtes représenté par un Y et vous parcourez le circuit en forme de labyrinthe en tournant dans le sens inverse des aiguilles d'une montre ; au passage vous modifiez le parcours et vous marquez des points.

Venant dans le sens opposé arrive un terrible monstre destructeur en forme de livre sterling, que vous devez à tout prix éviter en vous servant des touches 5 à 8 pour vous diriger dans le sens des flèches gravées sur ces touches.

Vous ne pouvez changer de couloir que lorsque vous arrivez devant une des ouvertures.

Faites attention car le monstre est intelligent et il n'est pas facile de l'esquiver.

Lorsque le monstre a réussi à vous détruire le score que vous avez obtenu est affiché ainsi que le score le plus élevé de la série. Il suffit de presser sur une touche pour recommencer une nouvelle partie.

58 CASSE BRIQUES (16 K)

Ce programme casse briques est intéressant pour deux raisons :

Premièrement c'est un jeu passionnant avec de nombreux niveaux de difficultés qui permet au joueur d'effectuer une multitude de parties sans se lasser en passant au niveau supérieur chaque fois que le niveau de difficulté inférieur est maîtrisé et que le score atteint régulièrement plusieurs centaines de points à chaque partie.

Deuxièmement dans ce programme, le langage machine n'est pas placé dans une instruction REM, comme pour les autres programmes que nous avons vus.

La partie du programme en langage machine reste dans la chaîne de caractères A$, du chargeur, jusqu'au moment du lancement du programme casse briques. Le langage machine est alors placé dans la mémoire RAM vers la fin de cette partie de la mémoire, à partir de l'adresse 30000, et non comme nous en avions l'habitude dans les premières cases disponibles de cette mémoire RAM.

Il faut cependant signaler que, pour un même programme machine, le fait de ne pas placer le programme machine dans une instruction REM, prend trois fois plus de place dans la mémoire RAM, ce qui est parfois un inconvénient majeur.

Voici 295 octets qui constituent le langage machine du programme casse briques.

```
2A 20 6E 3A 33 6E BC 30 0A 3A
32 6E BD 30 04 01 01 00 C9 16
00 1E 00 2A 22 6E AF BE 28 05
16 01 CD 4A 76 2A 24 6E BE 28
05 1E 01 CD 4A 76 BA 20 11 BB
20 0E ED 4B 26 6E 09 BE 28 2D
16 01 5A CD 4A 76 BA 28 0F 3A
26 6E ED 44 32 26 6E 3A 27 6E
2F 32 27 6E AF BB 28 0F 3A 28
6E ED 44 32 28 6E 3A 29 6E 2F
32 29 6E 18 11 2A 26 6E ED 4B
28 6E 09 ED 4B 20 6E C5 09 22
20 6E 2A 26 6E ED 4B 20 6E 09
22 22 6E 2A 28 6E 09 22 24 6E
7A 83 57 AF BA 20 07 E1 36 00
60 69 36 80 0E 0A 06 00 04 20
FD 0D 20 F8 3A 31 6E 47 AF B8
38 04 01 00 00 C9 3A 25 40 47
3E F7 B8 20 04 1E FD 18 07 3E
EF B8 20 4E 1E 03 3A 2A 6E F5
83 06 FE B8 20 02 3E 01 06 1F
B8 20 02 3E 1C 21 11 11 22 25
```

```
40 C1 48 F5 06 00 2A 2B 6E E5
09 70 23 70 23 70 2A 2D 6E E5
09 70 23 70 23 70 E1 D1 F1 D5
4F 32 2A 6E 16 80 09 72 23 72
23 72 E1 09 72 23 72 23 72 C3
30 75 3E FD B8 C2 30 75 01 01
00 C9 3E 08 BE 20 06 36 00 21
31 6E 35 AF C9
```

Le jeu de casse briques consiste à détruire un mur de briques à l'aide d'une balle qui rebondit constamment.

Chaque fois que la balle frappe une brique, celle-ci est détruite et un point est marqué.

Le joueur doit empêcher la balle de sortir du cadre en la renvoyant vers le mur de briques à l'aide d'une des raquettes. Il dispose de deux raquettes, en choisissant l'une ou l'autre des raquettes il est possible d'atteindre toutes les briques du mur.

Lorsque la balle traverse le mur, les briques sont détruites automatiquement jusqu'à ce que la balle ressorte.

En pressant la touche 0 les raquettes vont vers la droite et en pressant la touche 1 elles vont vers la gauche.

Voici le programme casse briques :

```
  10 FAST
  20 LET DFILE=PEEK 16396+256*PE
EK 16397
  30 LET A$="2A206E3A336EBC300A3
A326EBD300401010 0C916001E002A226
EAFBE2805 1601CD4A762A246EBE28051
E01CD4A76BA2011BB200EED4B266E09B
E282D16015ACD4A76BA280F3A266EED4
432266E3A276E2F32276EAFBB280F3A2
86EED4432286E3A296E2F32296E18112
A266EED4B286E09ED4B206EC5092220 6
E2A266EED4B206E0922226E2A286E092
2246E7A8357AFBA2007E13600606936 8
00E0A06000420FD0D20F83A316E47AFB
838040100 00C93A25404 73EF7B82004 1
EFD18073EEFB8204E1E033A2A6EF5830
6FEB820023E01061FB820023E1C21111
12225 40C148F506002A2B6EE50970237
023702A2D6EE509702370237 0E1D1F1D
54F322A6E16800972237223 72E109722
3722372C330753EFDB8C23075010100C
93E08BE20063600213 16E35AFC9"
  40 LET A=30000
  45 FOR B=1 TO LEN A$-1 STEP 2
  50 LET C=CODE A$(B)-28
  54 LET D=CODE A$(B+1)-28
  56 POKE A,16*C+D
```

```
  58 LET A=A+1
  60 NEXT B
  70 LET N=0
  71 CLS
  75 PRINT "QUEL NIVEAU ?","2-TR
ES BON",,"4-BON",,"8-MOYEN",,"D-
DEBUTANT"
  80 PAUSE 5000
  85 LET Z=(CODE INKEY$-28)*3+8
  90 POKE 30155,Z
 100 CLS
 110 LET A=28192
 120 LET B=INT (RND*30)+2+DFILE+
33*6
 130 POKE A,B-256*INT (B/256)
 135 POKE A+1,INT (B/255)
 140 POKE A+2,B+1-256*INT ((B+1)
/256)
 145 POKE A+3,INT ((B+1)/256)
 150 POKE A+4,B+33-256*INT ((B+3
3)/256)
 155 POKE A+5,INT ((B+33)/256)
 160 POKE A+6,1
 165 POKE A+7,0
 170 POKE A+8,33
 175 POKE A+9,0
 185 POKE A+10,16
 190 LET B=DFILE+33*21+1
 195 POKE A+11,B-256*INT (B/256)
 200 POKE A+12,INT (B/256)
 205 LET B=DFILE+33*14+1
 210 POKE A+13,B-256*INT (B/256)
 215 POKE A+14,INT (B/256)
 220 POKE A+17,90
 225 LET B=DFILE+33*14+3
 230 POKE A+18,B-256*INT (B/256)
 235 POKE A+19,INT (B/256)
 240 SLOW
 250 PRINT "    <-- 1 *CASSE BRIQ
UES* 0-->"
 260 PRINT "[redacted]"
 280 FOR X=0 TO 21
 290 PRINT AT X,0;"[redacted]";TAB 31;"[redacted]"
 300 NEXT X
 310 FOR Y=5 TO 7
 340 PRINT AT Y,1;"▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒
"
 350 NEXT Y
 360 PRINT AT 14,16;"[redacted]";AT 21,
16;"[redacted]"
 400 LET A=USR 30000
 410 IF A THEN GOTO 500
 420 LET N=N+1
 430 GOTO 100
 500 LET P=N*90+90-PEEK 28209
 510 PRINT AT 10,0;"VOUS AVEZ ";
P;" POINT";
 520 IF P>1 THEN PRINT "S"
```

```
530 PRINT AT 12,0;"AU NIVEAU ";
(Z-8)/3
540 PRINT AT 20,0;"APPUYEZ POUR
UN AUTRE ESSAI"
550 PAUSE 5000
560 GOTO 70
```

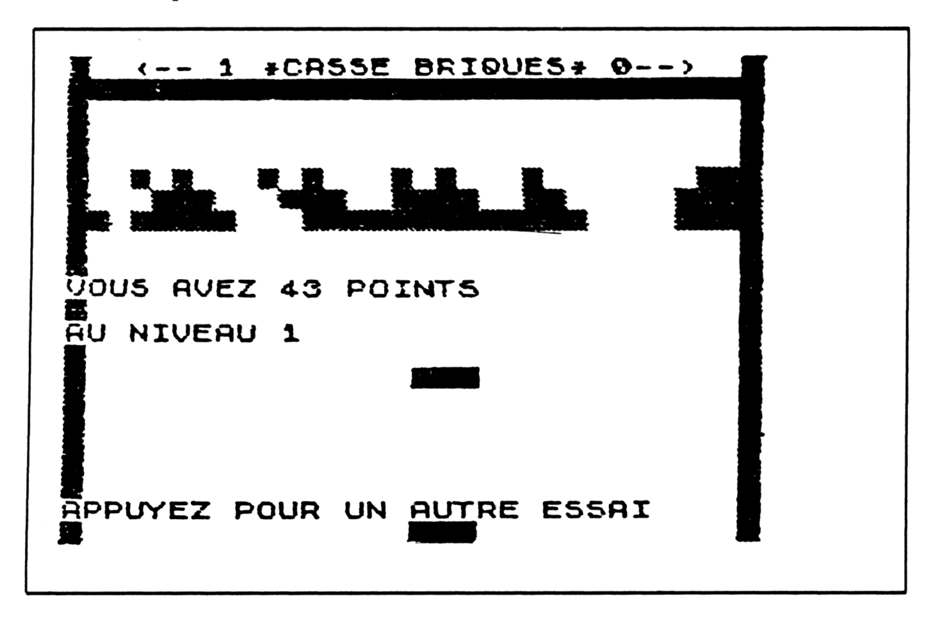
Lorsque le programme casse briques est lancé, le ZX 81 vous demande quel est le niveau de difficulté que vous choisissez, parmi 4 niveaux.

Vous pouvez choisir un niveau différent en choisissant un nombre hexadécimal entre 0 et F. (0 = champion, F = super débutant).

A la fin de la partie le score que vous avez atteint s'inscrit sur l'écran.

La partie continue tant que vous ne laissez pas sortir la balle du cadre. Lorsque le score de 90 points est atteint, le mur qui a presque disparu se reconstitue et vous continuez à augmenter votre score.

Voici un exemple de l'affichage obtenu sur l'écran, à la fin d'une partie de casse briques :



Il suffit de presser la touche NEW LINE pour recommencer une nouvelle partie.

Aux niveaux correspondants à un nombre inférieur à 5, la balle se déplace à grande vitesse ; il faut alors des réflexes très rapides pour l'intercepter à l'aide de la raquette.

Cette rapidité est obtenue par l'utilisation du langage machine.



NOUVEAUX CARACTERES GEANTS (16 K)

Nous avons vu un programme de caractères géants pour ZX 81 1 K dans les chapitres précédents. Ce programme sans utilité pratique se contentait d'afficher un caractère géant pendant quelques secondes sur l'écran. Ce programme avait pour but de montrer la différence de rapidité d'exécution entre un même programme écrit en langage machine et en langage Basic.

Celui que nous allons voir est plus utilitaire, car il affiche sur l'écran 6 lignes de caractères géants, chaque ligne comportant 8 caractères.
Voici les 136 octets de ce programme en langage machine.

```
2A 0C 40 23 22 0E 40 2A 0E 40
36 B1 CD BB 02 2C 20 FA CD BB
02 55 14 28 F9 E5 C1 CD 6D 07
21 00 1E C6 7D 5F 1A 17 17 17
CB 12 5F 19 0E 04 06 04 56 23
5E 23 E5 AF CB 12 1F CB 12 1F
CB 13 1F CB 13 1F 1F 1F 1F 1F
CB 5F 28 05 2F E6 0F C6 80 2A
0E 40 77 23 22 0E 40 10 DC 11
1D 00 2A 0E 40 19 22 0E 40 E1
0D 20 C6 11 80 00 2A 0E 40 A7
ED 52 22 0E 40 7E FE 76 20 8F
11 64 00 19 22 0E 40 ED 5B 10
40 ED 52 20 80 C9
```

Ce programme machine est placé dans la chaîne de caractères A$ du programme chargeur qui suit. Cette chaîne est construite en 3 temps.

```
    1 REM .......................
................................
................................
................................
................................
..........
   10 LET A=16514
   15 LET A$="2A0C4023220E402A0E4
036B1CDBB022C20FACDBB02551428F9E
5C1CD8D0721001EC67D5F1A171717"
   20 LET A$=A$+"CB125F190E040604
56235E23E5AFCB121FCB121FCB131FCB
131F1F1F1F1FCB5F28052FE60FC680"
   25 LET A$=A$+"2A0E407723220E40
10DC111D002A0E4019220E40E10D20C6
1180002A0E40A7ED52220E407EFE7620
8F1164001922 0E40ED5B1040ED522080
C9"
   30 FOR B=1 TO LEN A$-1 STEP 2
- - - - - - - - - - - - - - - - - -
```

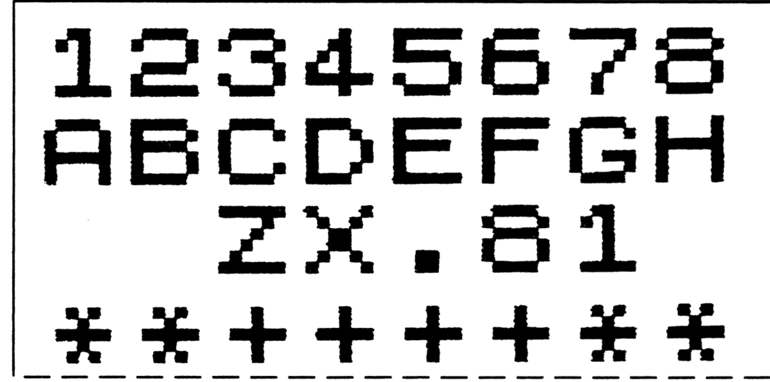
```
40 LET C=CODE A$(B)-28
50 LET D=CODE A$(B+1)-28
60 POKE A,16*C+D
70 LET A=A+1
80 NEXT B
90 STOP
```

Lorsque le programme chargeur a été lancé et que le langage machine est contenu dans l'instruction REM de la ligne 1, le programme définitif qui suit est complété ainsi :

```
    1 REM E£RND76:RNDE:RND■■LN ■■
G4 IF LN ■■?=C RAND FAST AT LN ■
■5 2LEN ??,***ACS >?;:■■■ ???? FA
ST ■ACS >3ACS >3ACS <3ACS <33333
ACS ?C■ J NEW ?LEN ■E:RND?76:RND(
>=)1 E:RND;6:RND LPRINT $4LEN )■
 E:RND■ GOSUB ?6:RND? ;6:RND GOS
UB ?(RND GOSUB ?4■TAN ..........
...............
   10 REM CARACTERES GEANTS
   20 RAND USR 16514
```

Lorsque le programme est lancé, le curseur apparaît en haut et à gauche de l'écran indiquant l'emplacement de la première lettre lorsque la touche correspondant à cette lettre sera pressée. Pressons quelques touches : à chaque fois un caractère géant vient se placer instantanément à côté du précédent.

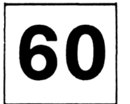
Lorsque les six lignes de caractères géants ont rempli l'écran, le programme revient au langage Basic. L'exemple suivant montre l'écran à ce moment.



```
37+5=42
```

Ce programme peut permettre de nombreuses applications scolaires par exemple pour servir de tableau et afficher des opérations arithmétiques.

Il peut aussi être placé en tête d'un programme pour établir des titres en caractères géants. Le texte à afficher peut être placé dans une chaîne de caractères par exemple B$, et l'affichage pourra ainsi être obtenu à tout moment grâce à PRINT B$.



60 L'EXPLORATEUR GALACTIQUE (1 K)

Ce programme montre qu'il est possible de créer des jeux intéressants et relativement complexes pour la version 1 K du ZX 81.

Avec ce programme vous explorez la galaxie à bord de votre astronef. Chaque fois que vous touchez un astéroïde représenté par un point, vous ajoutez 10 points à votre score. Chaque fois que vous touchez un système solaire représenté par un astérisque vous ajoutez 50 points à votre score.

Vous devez éviter les terribles nuages noirs de l'espace. Si vous heurtez un nuage noir, la partie est terminée car il vous retient captif.

Voici les huit octets de la portion en langage machine :

```
00 2A 0E 40 4E 06 00 C9
```

Le programme chargeur qui suit va placer ces octets dans l'instruction REM de la ligne 1.

```
 1 REM .........
10 LET A=16514
20 LET A$="002A0E404E0600C9"
30 FOR B=1 TO LEN A$-1 STEP 2
40 LET C=CODE A$(B)-28
50 LET D=CODE A$(B+1)-28
60 POKE A,16*C+D
70 LET A=A+1
80 NEXT B
```

La ligne 1 contenant la partie du programme en langage machine va se trouver en tête du programme astronef ci-dessous :

```
   1 REM  E:RND?▞ TAN .
  10 REM ASTRONEF
  20 LET S=0
  40 LET H=16
  60 LET H=H+(INKEY$="0")-(INKEY
$="1")
  70 PRINT AT 3,H;
  80 LET P=USR 16514
  90 LET S=S+(10 AND P=27)+(50 A
ND P=23)
```

```
 100 IF P=128 THEN GOTO 200
 110 PRINT ("V" AND INKEY$="")+(
"<" AND INKEY$="1")+(">" AND INK
EY$="0");AT 10,RND*31;CHR$ 27;AT
 10,RND*31;CHR$ 23 AND RND>.75;A
T 3,H;CHR$ 0;AT 10,RND*27;"███"
 120 SCROLL
 130 GOTO 60
 200 PRINT AT 3,H;"█";AT 18,14;"
SCORE ";S
```

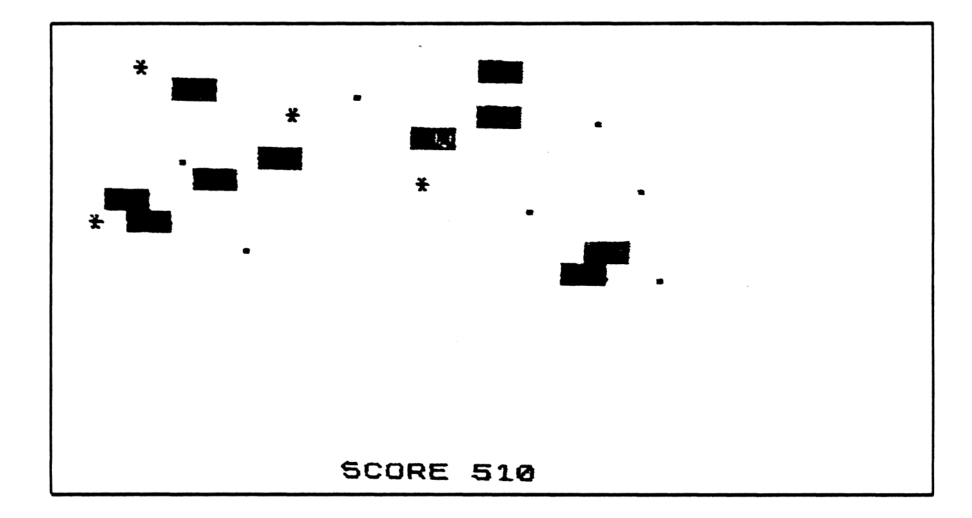
Lorsque ce programme est lancé, vous vous retrouvez aux commandes de votre astronef en train de parcourir la galaxie.

Vous disposez de la touche 1 pour diriger l'astronef vers la gauche et de la touche 0 pour le diriger sur la droite.

En vous servant de ces commandes vous devez toucher le maximum d'astéroïdes et de systèmes solaires et éviter les terribles nuages noirs.

Pour économiser de la mémoire, la ligne 110 regroupe une série de commandes PRINT qui devraient normalement être réparties sur plusieurs lignes.

L'exemple qui suit montre l'image sur l'écran à la fin de la partie, lorsque l'astronef vient d'être capturé par un nuage noir et que le score obtenu s'affiche sur l'écran.



ANNEXE

La liste suivante donne le jeu d'instructions du ZX 81 utilisé dans les programmes de cet ouvrage.

Caractère	Code hexa.	Mnémonique
space	00	nop
▘	01	ld bc,NN
▝	02	ld (bc),a
▀	03	inc bc
▖	04	inc b
▌	05	dec b
▞	06	ld b,N
▛	07	rlca
▒	08	ex af,af'
🮏	09	add hl,bc
🮎	0A	ld a,(bc)
"	0B	dec bc
£	0C	inc c
$	0D	dec c
:	0E	ld, c,N
?	0F	rrca
(	10	djnz DIS
)	11	ld de,NN
>	12	ld (de),a
<	13	inc de
=	14	inc d
+	15	dec d
−	16	ld d,N
*	17	rla
/	18	jr DIS
;	19	add hl,de
,	1A	ld a,(de)
.	1B	dec de
0	1C	inc e
1	1D	dec e
2	1E	ld e,N
3	1F	rra
4	20	jr nz,DIS
5	21	ld hl,NN
6	22	ld (NN),hl
7	23	inc hl
8	24	inc h
9	25	dec h
A	26	ld, h,N
B	27	daa
C	28	jr z,DIS
D	29	add hl,hl
E	2A	ld hl,(NN)
F	2B	dec hl
G	2C	inc l
H	2D	dec l
I	2E	ld,l,N
J	2F	cpl
K	30	jr nc,DIS
L	31	ld sp,NN
M	32	ld (NN),a
N	33	inc sp
O	34	inc (hl)
P	35	dec (hl)
Q	36	ld (hl),N
R	37	scf
S	38	jr c,DIS
T	39	add hl,sp
U	3A	ld a,(NN)
V	3B	dec sp
W	3C	inc a
X	3D	dec a
Y	3E	ld a,N
Z	3F	ccf
RND	40	ld b,b
INKEY$	41	ld b,c
PI	42	ld b,d
non utilisé	43	ld b,e
	44	ld b,h
	45	ld b,l
	46	ld b,(hl)
	47	ld b,a

Caractère	Code hexa.	Mnémonique
	48	ld c,b
	49	ld c,c
	4A	ld c,d
	4B	ld c,e
	4C	ld c,h
	4D	ld c,l
	4E	ld c,(hl)
	4F	ld c,a
	50	ld d,b
	51	ld d,c
	52	ld d,d
	53	ld d,e
	54	ld d,h
	55	ld d,l
	56	ld d,(hl)
	57	ld d,a
	58	ld e,b
non utilisé	59	ld e,c
	5A	ld e,d
	5B	ld e,e
	5C	ld e,h
	5D	ld e,l
	5E	ld e,(hl)
	5F	ld e,a
	60	ld h,b
	61	ld h,c
	62	ld h,d
	63	ld h,e
	64	ld h,h
	65	ld h,l
	66	ld h,(hl)
	67	ld h,a
	68	ld l,b
	69	ld l,c
	6A	ld l,d
	6B	ld l,e
	6C	ld l,h
	6D	ld l,l
	6E	ld l,(hl)
	6F	ld l,a
curseur ⇧	70	ld (hl),b
curseur ⇩	71	ld (hl),c
curseur ⇦	72	ld (hl),d

Caractère	Code hexa.	Mnémonique
curseur ⇨	73	ld (hl),e
GRAPHICS	**74**	**ld (hl),h**
EDIT	**75**	**ld (hl),l**
NEWLINE	**76**	**halt**
RUBOUT	77	ld (hl),a
K / L mode	78	ld a,b
FUNCTION	79	ld a,c
non utilisé	7A	ld a,d
non utilisé	7B	ld a,e
non utilisé	7C	ld a,h
non utilisé	7D	ld a,l
nombre	7E	ld a,(hl)
curseur	7F	ld a,a
█	80	add a,b
▟	81	add a,c
▙	82	add a,d
▄	83	add a,e
▜	84	add a,h
▌	85	add a,l
▚	86	add a,(hl)
▛	87	add a,a
▒	88	adc a,b
▓	89	adc a,c
▓	8A	adc a,d
inverse "	8B	adc a,e
inverse £	8C	adc a,h
inverse $	8D	adc a,l
inverse :	8E	adc a,(hl)
inverse ?	8F	adc a,a
inverse (	90	sub b
inverse)	91	sub c
inverse >	92	sub d
inverse <	93	sub e
inverse =	94	sub h
inverse +	95	sub l
inverse –	96	sub (hl)
inverse *	97	sub a
inverse /	98	sbc a,b
inverse ;	99	sbc a,c
inverse ,	9A	sbc a,d
inverse .	9B	sbc a,e
inverse 0	9C	sbc a,h
inverse 1	9D	sbc a,l

Caractère	Code hexa.	Mnémonique
inverse 2	9E	sbc a,(hl)
inverse 3	9F	sbc a,a
inverse 4	A0	and b
inverse 5	A1	and c
inverse 6	A2	and d
inverse 7	A3	and e
inverse 8	A4	and h
inverse 9	A5	and l
inverse A	A6	and (hl)
inverse B	A7	and a
inverse C	A8	xor b
inverse D	A9	xor c
inverse E	AA	xor d
inverse F	AB	xor e
inverse G	AC	xor h
inverse H	AD	xor l
inverse I	AE	xor (hl)
inverse J	AF	xor a
inverse K	B0	or b
inverse L	B1	or c
inverse M	B2	or d
inverse N	B3	or e
inverse O	B4	or h
inverse P	B5	or l
inverse Q	B6	or (hl)
inverse R	B7	or a
inverse S	B8	cp b
inverse T	B9	cp c
inverse U	BA	cp d
inverse V	BB	cp e
inverse W	BC	cp h
inverse X	BD	cp l
inverse Y	BE	cp (hl)
inverse Z	BF	cp a
""	C0	ret nz
AT	C1	pop bc
TAB	C2	jp nz,NN
non utilisé	C3	jp NN
CODE	C4	call nz,NN
VAL	C5	push bc
LEN	C6	add a,N
SIN	C7	rst 0
COS	C8	ret z
TAN	C9	ret

Caractère	Code hexa.	Mnémonique
ASN	CA	jp z,NN
ACS	CB	
ATN	CC	call z,NN
LN	CD	call NN
EXP	CE	adc a,N
INT	CF	rst 8
SQR	D0	ret nc
SGN	D1	pop de
ABS	D2	jp nc,NN
PEEK	D3	out N,a
USR	D4	call nc,NN
STR$	D5	push de
CHR$	D6	sub N
NOT	D7	rst 16
**	D8	ret c
OR	D9	exx
AND	DA	jp c,NN
<=	DB	in a,N
>=	DC	call c,NN
<>	DD	préfixe les instructions utilisant ix
THEN	DE	sbc a,N
TO	DF	rst 24
STEP	E0	ret po
LPRINT	E1	pop hl
LLIST	E2	jp po,NN
STOP	E3	ex (sp),hl
SLOW	E4	call po,NN
FAST	E5	push hl
NEW	E6	and N
SCROLL	E7	rst 32
CONT	E8	ret pe
DIM	E9	jp (hl)
REM	EA	jp pe,NN
FOR	EB	ex de,hl
GOTO	EC	call pe,NN
GOSUB	ED	
INPUT	EE	xor N
LOAD	EF	rst 40
LIST	F0	ret p
LET	F1	pop af
PAUSE	F2	jp p,NN
NEXT	F3	di

Caractère	Code hexa.	Mnémonique
POKE	F4	call p,NN
PRINT	F5	push af
PLOT	F6	or N
RUN	F7	rst 48
SAVE	F8	ret m
RAND	F9	ld sp,hl
IF	FA	jp m,NN
CLS	FB	ei
UNPLOT	FC	call m,NN
CLEAR	FD	préfixe les instructions utilisant iy
RETURN	FE	cp N
COPY	FF	rst 56

TABLE DES MATIÈRES

S.E.C.F.



EDITIONS RADIO

Langage machine pour ZX 81
P. Sirven

Service lecteurs

(à retourner à S.E.C.F.-Éditions Radio, 9, rue Jacob, 75006 Paris)

Pour nous permettre de vous proposer des ouvrages toujours meilleurs, nous souhaiterions recevoir vos critiques, appréciations et suggestions sur le présent livre :

____

____

____

____

Quels sont les ouvrages (thème, sujet, niveau) que vous souhaiteriez voir publier par notre société ?

____

____

____

____

Nous vous remercions de votre confiance et de votre coopération.

S.E.C.F.-Éditions Radio

Je désire recevoir gratuitement et sans engagement (mettre une croix dans la case) :

☐ Votre catalogue général (Electronique professionnelle et grand public, Informatique, Hi-Fi, Vidéo)
☐ Votre catalogue spécial informatique.

Nom : ____________________ Prénom : ____________________

Adresse : ____________________________________

____

Secteur d'activité et fonction : ____________________________

____

CENTRES D'INTÉRÊTS

☐ Electronique professionnelle
☐ Electronique de loisirs
☐ Vidéo
☐ Hifi, CB...
☐ Micro-informatique professionnelle
☐ Micro-informatique de loisirs
☐ Autres : ....
____

Voir au dos "Correspondance Auteurs"



Langage machine pour ZX 81
P. Sirven



Service lecteurs

Correspondance auteurs

(à retourner à S.E.C.F.-Éditions Radio, 9, rue Jacob, 75006 Paris)

Pour toute demande d'éclaircissements techniques relatifs à ce livre, formulez ci-dessous vos questions, avec le maximum de précisions :

Nom : ______________________ Prénom : ______________________

Adresse : ______________________

LANGAGE MACHINE

POUR ZX 81

Découvrez le langage machine et ses instructions sur le microprocesseur Z80. Pour cela, l'auteur explique comment introduire ces instructions dans votre ZX81 à l'aide du Basic.
60 programmes dont 10 jeux complets, accompagnés d'exemples d'utilisation, illustrent les différentes notions étudiées.

Un excellent moyen de gagner de précieux octets, d'accroître la vitesse d'exécution de vos programmes et donc d'en multiplier les performances.



ISBN 2 7091 0946 8
Code 90



LANGAGE MACHINE POUR ZX 81

P. SIRVEN